HOME E PERSONAL COMPUTER

Rita Bonelli

Commodore 128 oltre il manuale

GRUPPO EDITORIALE
JACKSON

Commodore 128 oltre il manuale

Rita Bonelli

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

© Copyright per la prima edizione:
Gruppo Editoriale Jackson - Giugno 1986
IIª RISTAMPA: Gennaio 1988

COORDINAMENTO EDITORIALE: Emma Bennati

COPERTINA: Silvana Corbelli

GRAFICA E IMPAGINAZIONE: Dino Bortolossi

FOTOCOMPOSIZIONE: Graphotek S.r.l. - Via Astesani, 16 - Milano

STAMPA: Rotolito Lombarda - Cologno Monzese - MI

Tutti i diritti sono riservati. Stampato in Italia. Nessuna parte di questo libro puó essere riprodotta, memorizzata in sistemi di archivio, o trasmessa in qualsiasi forma o mezzo, elettronico, meccanico, fotocopia, registrazione o altri senza la preventiva autorizzazione scritta dell'editore.

Commodore BASIC 7.0 Copyright 1985 Commodore Italiana S.p.a.
Copyright 1977 Microsoft.

CP/M Plus Versione 3.0 Copyright 1982 Digital Research, Inc..

Perfect Filer, Perfect Writer e Perfect Calc sono marchi registrati
Copyright 1985 Commodore Electronics Limited
Copyright 1984 THORN EMI Computer Software Inc..

INDICE

INTRODUZIONE V

Capitolo 1 - INTRODUZIONE ALLA PROGRAMMAZIONE
1.1 Il Sistema Operativo 1
1.2 I linguaggi simbolici 2
1.3 Le strutture fondamentali 5
1.4 La gestione delle periferiche 9
1.5 Il linguaggio macchina 11
1.6 Dal problema al programma 12

Capitolo 2 - PROGRAMMARE IN BASIC 7.0
2.1 Ingresso e uscita dei dati con tastiera e video 15
2.2 Istruzioni condizionali e cicliche 31
2.3 Sottoprogrammi e funzioni 33
2.4 Video a 80 colonne in modo testo 37
2.5 Video a 40 colonne in modo testo 42
2.6 Grafica con video a 40 colonne 44
2.7 Grafica con video a 80 colonne 55
2.8 I joystick 59
2.9 Il movimento e l'animazione delle figure 63
2.10 La stampante 79
2.11 I file su cassetta e su disco 80

Capitolo 3 - UTILIZZO DELLA MEMORIA
3.1 Gestione della memoria 81
3.2 Mappe della memoria 92
3.3 Utilizzo della memoria nei programmi in BASIC 114
3.4 Overlay di programmi 130

Capitolo 4 - ASSEMBLER E LINGUAGGIO MACCHINA
4.1 Programmazione in linguaggio macchina 133
4.2 Le istruzioni del microprocessore 8502 136
4.3 Utilizzo del MONITOR 163
4.4 Le routine del sistema e il loro utilizzo 167
4.5 Le note musicali 173

Capitolo 5 - IL MODO CP/M
5.1 Introduzione 175
5.2 I pacchetti integrati Perfect Filer, Writer e Calc 176

L'autrice ringrazia i dirigenti della Commodore Italiana S.p.a., per aver messo a disposizione le apparecchiature e la documentazione necessarie alla realizzazione dell'opera.

Un ringraziamento va anche ai signori Diego Perini e Marco Zampieri per le informazioni cortesemente fornite riguardo al chip 8563.

I programmi contenuti nel libro sono stati tutti provati utilizzando un calcolatore COMMODORE 128 collegato con il Monitor 1701 (che puó visualizzare anche a 80 colonne in monocromatico), l'unità a floppy disk 1571 e la stampante MPS 801 della COMMODORE.

INTRODUZIONE

Volendo parlare di questo nuovo calcolatore che la COMMODORE ci ha messo a disposizione possiamo cominciare con la frase *tre in uno*; infatti possiamo decidere ogni volta, scegliendo il modo, con quale dei tre calcolatori disponibili lavorare servendoci dell'elegante tastiera.

Chi è già un esperto del COMMODORE 64 puó scegliere il modo 64 e continuare a utilizzare tutti i programmi che possiede. Infatti il COMMODORE 128 è completamente compatibile; esso puó essere collegato con le stesse periferiche e utilizzato come un COMMODORE 64.

Il modo 128 ci mette a disposizione un calcolatore molto potente con 128K di memoria ed una nuova implementazione del BASIC, la versione 7.0. Inoltre sono disponibili periferiche più versatili come le unità a disco 1570 e 1571 e il monitor RGB a 80 colonne. Il COMMODORE 128 puó essere utilizzato con le periferiche del COMMODORE 64, anche se con queste non si possono sfruttare in pieno le nuove possibilità. E' possibile collegare contemporaneamente un monitor a 40 colonne ed uno a 80 e visualizzare su ambedue.

Infine il modo CP/M ci permette di lavorare con un calcolatore compatibile sul quale puó essere utilizzata la vasta gamma dei programmi CP/M disponibili.

Lo scopo di questo libro non è quello di ripetere il contenuto della GUIDA INTRODUTTIVA e del MANUALE DI SISTEMA venduti nella traduzione in italiano insieme al calcolatore, ma piuttosto fornire un aiuto per un approccio sistematico alla programmazione e alcune notizie aggiuntive sul sistema.
Lo scopo che ci sta più a cuore è quello di suggerire un metodo con il quale avvicinarsi a un personal e entrare con consapevolezza nel mondo dell'informatica. Attualmente il mercato ci offre dei calcolatori molto potenti ad un prezzo abbastanza accessibile e l'utente si trova a contatto con questi strumenti senza avere avuto occasione prima di occuparsi dell'argomento. Il nostro libro non vuole essere un trattato di informatica, ma piuttosto contribuire in un modo diciamo *globale* alla conoscenza di un calcolatore. Il lettore potrà ricevere un aiuto nell'apprendimento dell'uso della sua macchina e degli stimoli per approfondire alcuni argomenti. Noi riteniamo che il modo più soddisfacente per avvicinarsi all'informatica è quello di avere a disposizione un calcolatore, soprattutto se è facile da usare come un personal e, in particolare, come il COMMODORE 128.

Non ci occupiamo del modo 64, infatti sul COMMODORE 64 esiste una vasta letteratura, alla quale noi stessi abbiamo dato un contributo con i tre libri: COMMODORE 64 IL BASIC, I FILE e LA GRAFICA E IL SUONO, pubblicati dal Gruppo Editoriale Jackson, ai quali rimandiamo.

CAPITOLO 1

INTRODUZIONE ALLA PROGRAMMAZIONE

1.1 IL SISTEMA OPERATIVO

Il Sistema Operativo è un insieme di programmi che consentono di utilizzare in modo semplice le risorse del calcolatore. Nel COMMODORE 128 sono residenti nella memoria ROM (Read Only Memory) due Sistemi Operativi, uno di essi viene utilizzato nel modo 64, l'altro nel modo 128.
In questi due modi possono essere caricati dal dischetto alcuni programmi di utilità che arricchiscono le possibilità del calcolatore, ma non sono indispensabili. Per poter utilizzare il COMMODORE 128 con il Sistema Operativo CP/M, che non è residente, è necessario invece caricare nella memoria RAM (Random Access Memory, memoria per lettura e scrittura) dall'apposito dischetto il CP/M Plus 3.0.

Quando si accende il calcolatore, senza predisporre un particolare modo, esso funziona nel modo 128. Dopo aver eseguito tutti i controlli interni necessari, viene visualizzato il messaggio iniziale e il calcolatore resta in attesa di comandi, in *stato sistema*. In tale stato è attivo l'EDITOR che consente di utilizzare il calcolatore in *modo testo* comunicando con esso tramite le due principali periferiche, che sono la tastiera e il video. In questo stato è attivo il set di caratteri maiuscolo/grafico, si possono scrivere ed eseguire comandi del sistema, scrivere ed eseguire in immediato istruzioni BASIC 7.0, oppure memorizzare un programma in BASIC 7.0 per esecuzione differita. Le istruzioni BASIC vengono eseguite in *stato programma*, il quale permane fino a quando si verificano alcuni tipi di errore che interrompono il programma, o il programma termina, o l'operatore interviene dall'esterno per interrompere il programma.

In stato programma è attivo il programma interprete BASIC, che provvede ad interpretare ed eseguire le istruzioni BASIC. L'interprete BASIC per lavorare utilizza molte routine del sistema.

Il Sistema Operativo gestisce tutte le risorse hardware disponibili in base ad un meccanismo di interruzioni prioritarie che è stato organizzato tenendo conto delle caratteristiche interne del calcolatore e di quelle delle interfacce con le periferiche.

Le principali funzioni del Sistema Operativo del COMMODORE 128 sono:
- gestione del video chip a 40 colonne;
- gestione del video chip a 80 colonne;
- gestione della tastiera;
- EDITOR di schermo;
- gestione del SID, chip sintetizzatore del suono;
- gestione dei chip CIA di I/O;

– gestione dei banchi di memoria;
– uso del MONITOR.
Nel capitolo 3 è descritto l'utilizzo della memoria nei tre modi di funzionamento del calcolatore.

Quando si programma in BASIC non è necessario studiare a fondo il Sistema Operativo, è necessario solo conoscere i comandi e saperli usare. Inoltre, dato che il BASIC 7.0 è estremamente ricco di istruzioni, molte operazioni che nel modo 64 possono essere svolte solo programmando in linguaggio macchina, nel modo 128 possono invece essere svolte in BASIC.

Nel modo 64 il calcolatore risulta a tutti gli effetti un COMMODORE 64 ed è possibile utilizzare tutto il software per esso disponibile. A parte questo è indubbiamente più soddisfacente lavorare nel modo 128, infatti in tale modo si dispone di un calcolatore molto più potente.

Il modo CP/M consente di prendere confidenza con questo tipo di Sistema Operativo e di utilizzare alcuni programmi di utilità presenti sui dischetti forniti, ma è necessario acquistare dei programmi adatti per lavorare. Si possono acquistare programmi di linguaggio come: interprete BASIC, compilatore COBOL, compilatore FORTRAN, e altri, e pacchetti di programmi dedicati già pronti per l'uso.

Nel modo 128 il calcolatore puó essere programmato in BASIC 7.0 o nel linguaggio macchina del processore 8502. L'utente puó scrivere programmi in linguaggio macchina, utilizzare routine del Sistema Operativo chiamandole direttamente, e in questo tipo di lavoro si puó giovare delle possibilità offerte dal MONITOR.
Lo stesso discorso vale per il modo 64. Per poter programmare in linguaggio macchina nel modo CPM è necessario utilizzare il processore Z80, che ha un linguaggio macchina diverso da quello del processore 8502.

1.2 I LINGUAGGI SIMBOLICI

I linguaggi simbolici di programmazione consentono di programmare il calcolatore in modo più semplice che non utilizzando il linguaggio macchina. Essi si basano su insiemi di regole studiate per soddisfare determinate esigenze e tali da consentire una traduzione automatica, cioè da parte di un programma, nel linguaggio macchina di un particolare calcolatore.

Il livello di simbolismo puó essere più o meno elevato; abbiamo:
– i linguaggi simbolici a basso livello, chiamati ASSEMBLER, che consentono di programmare utilizzando istruzioni simboliche e operandi simbolici, in generale con corrispondenza uno a uno tra istruzione simbolica e istruzione in linguaggio macchina;
– i linguaggi simbolici ad alto livello nei quali ad una istruzione simbolica corrispondono più istruzioni in linguaggio macchina.
Per i primi il programma traduttore si chiama ASSEMBLATORE. Per i secondi la fase di traduzione puó avvenire in due modi:
– prima dell'esecuzione del programma, e in questo caso il programma traduttore si chiama COMPILATORE;
– durante l'esecuzione del programma, e in questo caso il programma traduttore si chiama INTERPRETE.

Il BASIC è un linguaggio simbolico ad alto livello che viene interpretato, cioè la traduzione del programma avviene durante la sua esecuzione. La preparazione di un programma in BASIC risulta molto più semplice che non con un linguaggio che richiede la fase di compilazione. Infatti in BASIC si ha un costante controllo del programma con la possibilità di correggere gli errori immediatamente e riprovare, si lavora cioè in modo interattivo e l'utente ha la piacevole sensazione di poter colloquiare con il calcolatore.
Indubbiamente l'esecuzione di un programma compilato risulta più veloce di quella di un programma interpretato. In conseguenza un ottimo ambiente di lavoro è quello nel quale si dispone di un normale BASIC interpretato per preparare i programmi e della versione con compilatore per la traduzione finale dopo la messa a punto e il collaudo dei programmi. Questa situazione ideale si ha con il COMMODORE 64, per il quale esiste un buon compilatore per il BASIC 2.0. Ci auguriamo che sia presto disponibile sul mercato un buon compilatore anche per il BASIC 7.0 del COMMODORE 128.

Il BASIC è un linguaggio che si presta bene per affrontare i più vari tipi di problemi, consente una buona gestione delle periferiche, ed inoltre non risulta difficile utilizzare nei programmi routine scritte in linguaggio macchina.
Nel COMMODORE 128 la presenza del MONITOR in ROM facilita la preparazione delle routine in linguaggio macchina.

Il BASIC 7.0 è una implementazione molto ricca e consente di programmare in modo semplice. I dati da elaborare possono essere memorizzati in variabili singole e con indice; i tipi disponibili sono:
– numeriche intere, il cui nome termina con il suffisso '%' e che possono contenere numeri interi appartenenti all'intervallo -32768...+32767;
– numeriche non intere (chiamate anche reali), il cui nome non viene identificato da un particolare suffisso, per i numeri in formato esponenziale (floating point) con la precisione di circa 9 cifre decimali, utilizzando un byte per l'esponente e 4 byte per la mantissa (cifre significative);
– alfanumeriche, il cui nome termina con il suffisso '$', per memorizzare fino a 255 caratteri;
– variabili con indice dei tre tipi validi per le variabili singole; per le quali il numero di dimensioni puó arrivare a 255.

Inoltre il BASIC puó accedere direttamente ad indirizzi assoluti di memoria; nei byte corrispondenti puó memorizzare dati e da essi puó riprenderli.

Le istruzioni disponibili possono essere ripartite nei seguenti gruppi:
– *assegnazione*, che consentono di trasferire valori nelle variabili;
– *controllo*, che consentono di analizzare una condizione, e scegliere una strada in base al risultato trovato;
– *esecuzione cicli*, che consentono di eseguire ripetutamente sequenze di istruzioni in base al verificarsi di particolari condizioni;
– *comunicazione con le periferiche*, che consentono di ricevere dati dall'esterno e di comunicare dati all'esterno, gruppo che comprende anche le istruzioni per la grafica e il suono;
– *servizio*, che consentono di eseguire operazioni come azzeramento delle variabili, pulizia della memoria e simili;
– *funzioni*, gruppo molto vasto che comprende funzioni di diverso tipo. La differenza tra funzione e istruzione sta nel fatto che una funzione non è autonoma, ma fornisce un risultato, e quindi viene utilizzata da una istruzione.

I *sottoprogrammi* sono costituiti da sequenze di istruzioni che svolgono uno specifico compito; essi possono essere richiamati da diversi punti del programma e, al termine della loro esecuzione, restituiscono il controllo al programma nel punto successivo al richiamo. Deve esistere in conseguenza un'istruzione per il richiamo del sottoprogramma e un istruzione di uscita dal sottoprogramma.
L'utilizzo di sottoprogrammi risulta molto utile per evitare di ripetere più volte la stessa sequenza di istruzioni, evita gli errori, infatti se il sottoprogramma funziona si è sicuri dei risultati, e inoltre contribuisce alla chiarezza della stesura del programma. Si veda la figura 1.1.

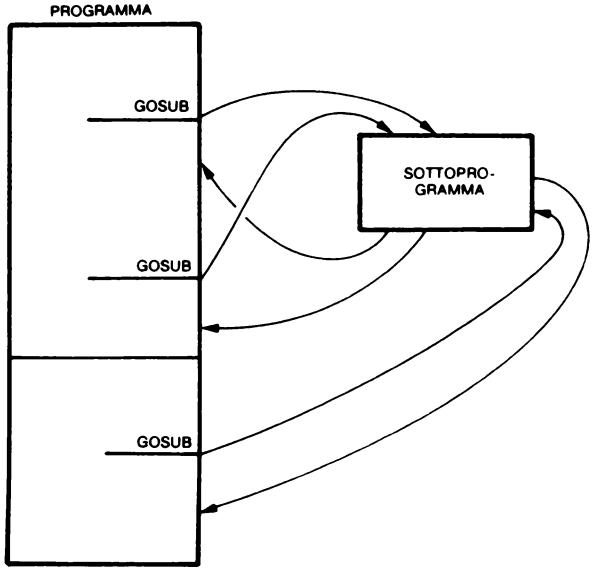


Fig. 1.1 Chiamata di un sottoprogramma

Un sottoprogramma è esterno al programma quando non ne fa parte, interno quando ne rappresenta una parte. Il BASIC consente di scrivere sottoprogrammi interni al programma, che utilizzano le stesse variabili del programma principale, o variabili proprie, opportunamente definite, senza particolari convenzioni per il passaggio dei dati. Sia i dati di ingresso che quelli di uscita sono trattati con i nomi delle variabili che li contengono, accessibili al programma principale e ad eventuali altri sottoprogrammi.
Il BASIC puó utilizzare sottoprogrammi in linguaggio macchina con diverse possibilità di comunicazione per i dati; trattiamo questo argomento nel capitolo 4.

1.3 LE STRUTTURE FONDAMENTALI

Le istruzioni di un programma scritto in un qualunque linguaggio presentano dal punto di vista della struttura delle caratteristiche comuni. Per prima cosa il programma puó essere scomposto in parti che hanno un solo punto di entrata e un solo punto di uscita. Analizzando tutte queste parti si scopre che esse appartengono alle strutture fondamentali che elenchiamo.

STRUTTURA SEQUENZIALE
Una sequenza di istruzioni che vengono eseguite una dopo l'altra; essa è schematizzata nel diagramma della figura 1.2.

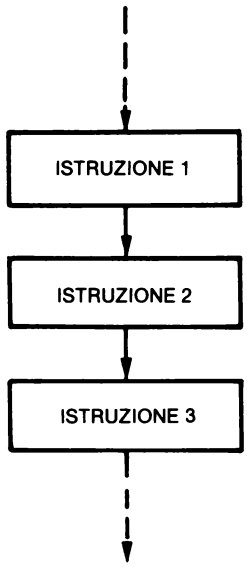
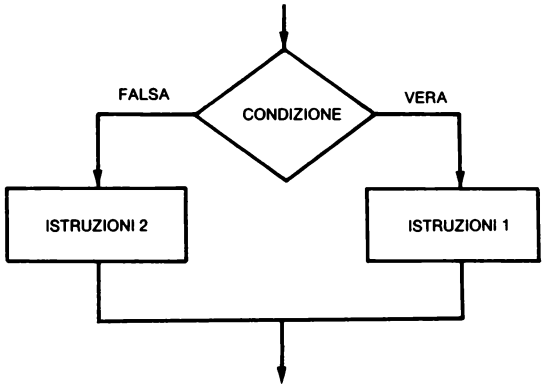


Fig. 1.2 Struttura sequenziale

STRUTTURA CONDIZIONALE
Consiste nell'esame di un'espressione condizionale e nel proseguimento del programma da uno tra due punti alternativi in dipendenza dal valore dell'espressione condizionale esaminata. La struttura puó essere schematizzata nella figura 1.3.

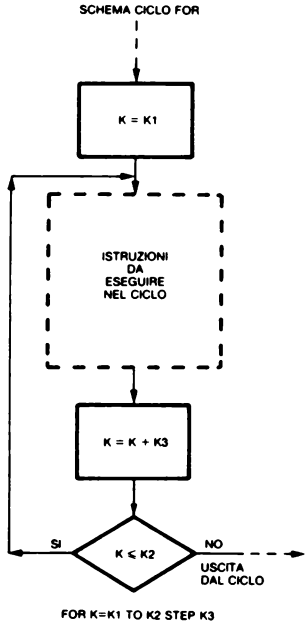
STRUTTURA CICLICA
Consiste nell'esecuzione ripetuta di una sequenza di istruzioni, o un numero prefissato di volte, o un numero di volte che dipende dal verificarsi o meno di una condizione esaminata.
Riportiamo nella figura 1.4 una struttura ciclica controllata in modo enumerativo, che corrisponde all'istruzione FOR del BASIC 7.0, e nelle figure 1.5 e 1.6 altri tipi



IF CONDIZIONE THEN ISTRUZIONI 1: ELSE ISTRUZIONI 2

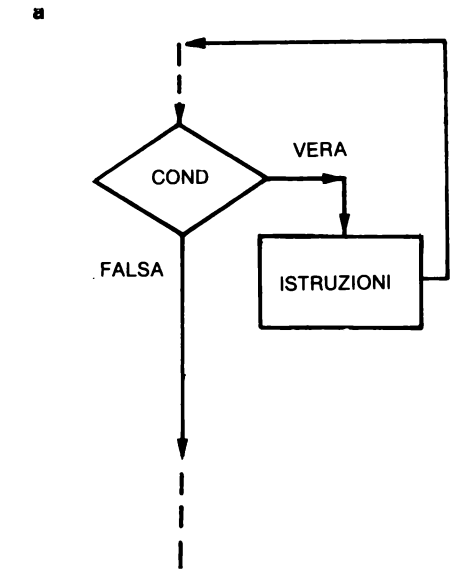
Fig. 1.3 Struttura condizionale

di strutture cicliche, che corrispondono alle altre istruzioni per il controllo dei cicli implementate nel BASIC 7.0.

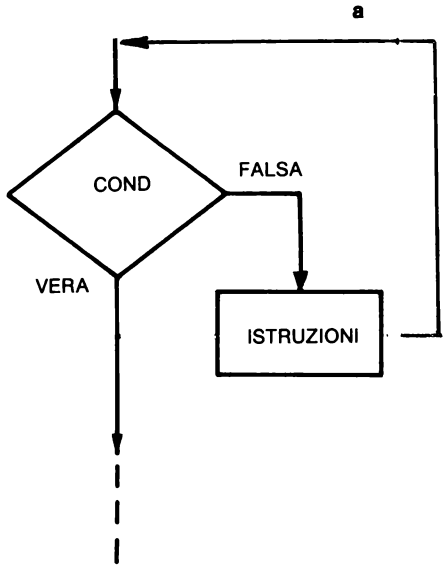


FOR K=K1 TO K2 STEP K3

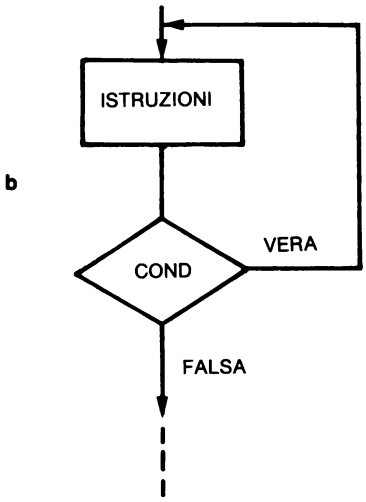
Fig. 1.4 Struttura ciclica FOR



DO WHILE COND: ISTRUZIONI: LOOP

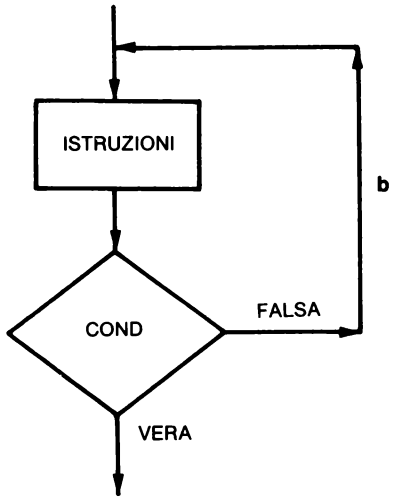


DO UNTIL COND: ISTRUZIONI: LOOP



DO:ISTRUZIONI: LOOP WHILE COND

Fig. 1.5 Struttura ciclica DO...WHILE

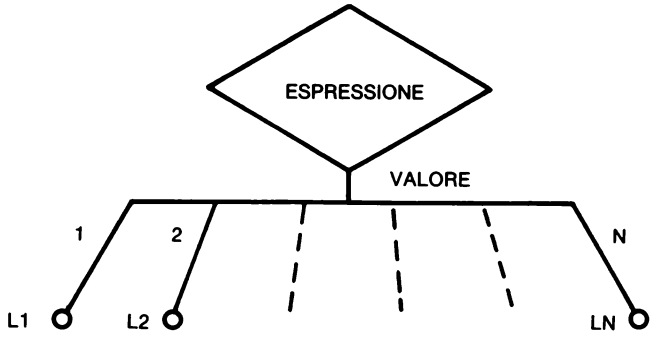


DO: ISTRUZIONI: LOOP UNTIL COND

Fig. 1.6 Struttura ciclica DO...UNTIL

Si noti che la struttura ciclica della figura 1.4 viene percorsa almeno una volta. Nella figura 1.5 la struttura rappresentata nella parte *a* puó non essere mai percorsa se la condizione è a priori falsa, mentre quella rappresentata nella parte *b* viene comunque percorsa almeno una volta. Nella figura 1.6 la struttura rappresentata nella parte *a* puó non essere mai percorsa se la condizione risulta a priori vera, mentre quella rappresentata nella parte *b* viene comunque percorsa almeno una volta. Si noti inoltre che il costrutto UNTIL...COND corrisponde a WHILE...NOT COND.

STRUTTURA A SCELTA MULTIPLA
Consente di operare la scelta tra più strade (non solo tra due come la STRUTTURA CONDIZIONALE) a seconda dei valori interi assunti da un'espressione. Schematizziamo la struttura nella figura 1.7; essa corrisponde all'istruzione ON...GOTO (o GOSUB) del BASIC 7.0.



ON ESPRESSIONE GO TO L1, L2, .., LN

Fig. 1.7 Struttura a scelta multipla

In generale i linguaggi di programmazione mettono a disposizione apposite istruzioni per realizzare le ultime tre strutture, e noi abbiamo fatto cenno a quelle disponibili nel BASIC 7.0. La prima struttura non viene realizzata da una istruzione specifica, ma da qualunque sequenza di istruzioni.

Oltre alle strutture viste esiste l'istruzione di *salto incondizionato*, realizzata in BASIC dall'istruzione GOTO, che consente di abbandonare una sequenza di programma e proseguire da un altro punto. Per una buona programmazione si raccomanda di fare un uso limitato di questa istruzione, infatti essa puó rendere poco leggibili i programmi, producendo degli intrecci di difficile comprensione.

Un programma deve essere soprattutto chiaro; infatti solo se si seguono delle regole di ordine si potrà intervenire a modificare il programma in seguito. In questo libro noi esponiamo programmi in BASIC e in ASSEMBLER; ambedue questi linguaggi consentono di scrivere programmi poco chiari e contorti, ma noi cerchiamo

di non farlo. Una buona regola da seguire è quella di definire le variabili e elencarle in una lista da tenere a portata di mano mentre si scrive il programma. Inoltre si deve evitare di scrivere programmi formati da un blocco unico; un buon programma dovrebbe avere la struttura di un blocco principale che utilizza richiamandoli una serie di sottoprogrammi, ciascuno con un compito specifico.
Per la chiarezza conta anche il modo di scrittura delle istruzioni. Per esempio il BASIC 7.0 non impone di utilizzare spazi separatori tra le parole chiave e gli altri elementi di un'istruzione; noi li usiamo ottendendo un listato di programma molto più leggibile. Uno spazio significa un byte in più in memoria, ma fortunatamente il COMMODORE 128 dispone di molta memoria. Questo ci consente anche di abbondare in frasi di commento che contribuiscono a rendere comprensibile il programma.
Purtroppo il mini-assembler utilizzabile con il MONITOR non ci permette di inserire i commenti; quando prepariamo routine in questo linguaggio dobbiamo scrivere a parte tutti i commenti necessari e il significato delle variabili utilizzate e conservare il tutto come documentazione.

1.4 LA GESTIONE DELLE PERIFERICHE

Le periferiche sono individuate dal calcolatore mediante un numero, chiamato DN (Device Number, numero dispositivo); tale numero puó essere preventivamente prefissato, o modificabile agendo su appositi switch esterni o interni alla periferica. Per il COMMODORE 128 abbiamo i seguenti valori:

Periferica	DN	Tipo operazioni consentite
Tastiera	0	Input
Registratore	1	Input/Output
RS-232	2	Input/Output
Video	3	Input/Output
Stampante	4/7	Output
Disco	8/11	Input/Output

L'insieme dei dati scambiati tra calcolatore e periferica si chiama *flusso* o *stream*. A seconda delle periferiche il flusso puó permanere oltre lo scambio con il calcolatore, questo dipende dal mezzo che fa da supporto. Il flusso di dati da tastiera non si conserva, quello verso la stampante si, come pure quello verso un floppy disk. Per il calcolatore ogni flusso di dati costituisce un *file* ed è contraddistinto da un numero, chiamato LFN (Logical File Number, numero logico del file), che serve ad individuare un particolare file tra tutti quelli gestiti. Lo LFN puó variare da 0 a 255, ma per il sistema il numero massimo di file gestibili contemporaneamente è dieci.
Con una periferica come l'unità a floppy disk singolo possono essere trattati contemporaneamente più file, per tutti risulta DN=8 mentre lo LFN è diverso per ogni file.

La tastiera e il video sono le periferiche principali del sistema e possono essere trattate diversamente dalle altre, cioè per comunicare con esse non è necessario eseguire tutte le operazioni altrimente richieste, ma basta scrivere in un modo particolare le istruzioni di input/output, modo che presuppone per default il DN di

queste periferiche e non richiede la dichiarazione di uno LFN. Ció non impedisce peró di gestirle come tutte le altre periferiche; nel capitolo 2 abbiamo riportato degli esempi al riguardo.

Un file puó essere organizzato in due modi diversi:
– una sequenza di caratteri consecutivi, ciascuno dei quali riflette il contenuto di un byte, con una segnalazione finale di fine file, che puó essere trasferito da e nella memoria del calcolatore in blocco (file binario);
– una sequenza di registrazioni che ha senso considerare singolarmente come una unità di informazione, ciascuna formata da un gruppo di dati (file di dati). Ogni registrazione si chiama *record* e puó essere trasferita da sola, tutta o in parte, da e nella memoria del calcolatore. I dati contenuti in un record si chiamano *campi* (field). Questo record si chiama più propriamente *record logico* in quanto fa riferimento a dati collegati tra loro per ragioni che dipendono dalla natura dell'argomento trattato.

Ogni supporto fisico di registrazione di un file impone delle limitazioni dovute alla sua natura; per esempio sulla carta di determinate dimensioni non entrano più di 80 caratteri per riga, se vogliamo stampare un record che contiene 140 caratteri dobbiamo ripartirlo su due righe. In questo caso possiamo parlare di *record fisico* per la riga del foglio di carta e di *record logico* per i dati che vogliamo stampare. Non sempre l'utente deve preoccuparsi della gestione dei record fisici, in quanto ci pensano le routine di input/output del sistema. Per esempio, quando inviamo dei dati in uscita al video ed il cursore è arrivato in fondo alla riga, esso va a capo automaticamente, ma se vogliamo una presentazione più personalizzata dei dati dobbiamo pensare noi a provocare un *vai a capo* al momento giusto.

Un altro elemento che serve a caratterizzare un flie è lo SA (Secondary Address, indirizzo secondario); esso puó avere valore da 0 a 15 e significato che varia da una periferica all'altra.
Inoltre per alcune periferiche è necessario assegnare un nome al file, lo FN (File Name), che è una stringa di caratteri alfanumerici.

Il BASIC mette a disposizione due gruppi di istruzioni per la gestione dei file; un gruppo riguarda i file organizzati in record e l'altro i file organizzati in caratteri (byte).

Le operazioni per la gestione dei file organizzati in record, che sono classicamente i file di dati, consistono in:
– apertura del file, con precisazione dei necessari parametri in modo da stabilire la comunicazione con la periferica e creare un'apposita entrata nella tabella per la gestione dei file;
– lettura o scrittura dei dati;
– chiusura del file con eliminazione dell'entrata dalla tabella dei file e della comunicazione con la periferica.
La tabella per la gestione dei file puó contenere le entrate di 10 file; essa conserva per ogni file tre elementi: LFN, DN e SA.

Le operazioni relative all'altro gruppo di file consentono di trasferire tutto il file o in input o in output. A questo gruppo appartengono i file di programmi sia in BASIC che in linguaggio macchina.

Inoltre in BASIC sono disponibili un gruppo di operazioni di servizio per la gestione dei file.

Le periferiche possono essere di due categorie diverse, quelle che non dispongono di un processore interno che contribuisce alla loro gestione e quindi sono completamente sotto il controllo del calcolatore, come per esempio il registratore a cassetta, e quelle che hanno un processore interno. A questa seconda categoria appartengono l'unità a floppy disk e le stampanti; in questo caso il calcolatore invia alla periferica un comando che viene utilizzato e sviluppato per il successivo scambio di dati.
La stampante ad aghi, per esempio, riceve i codici dei caratteri e li trasforma nella rappresentazione per punti dei caratteri, oppure riceve un comando che la fa passare in modo grafico e le fa interpretare diversamente i codici dei dati che riceve.
Quindi le istruzioni per la gestione dei file, pur avendo delle caratteristiche comuni, vanno studiate in modo particolare per ogni tipo di periferica.

1.5 IL LINGUAGGIO MACCHINA

Il linguaggio macchina nel modo 128 è quello del microprocessore 8502. Il MONITOR consente di scrivere i programmi in un ASSEMBLER ridotto, ma sufficiente a renderne più facile la preparazione. Esso inoltre consente di ricavare il codice ASSEMBLER da routine in linguaggio macchina già presenti in memoria, ed ha altre utili prestazioni. Un'istruzione in linguaggio macchina è formata da un codice operativo, che occupa un byte, e da un operando, che in certi casi manca (è implicito nel codice operativo), e che, se presente, puó occupare uno o due byte. L'operando puó essere un dato, per esempio un numero da sommare, o l'indirizzo di un dato o di un'altra istruzione. Gli indirizzi possono essere ottenuti in diversi modi, come si puó vedere nel capitolo 4, che tratta la programmazione in ASSEMBLER e in linguaggio macchina.

In questo paragrafo facciamo brevi cenni sulla rappresentazione delle informazioni nella memoria del calcolatore. L'unità elementare di memoria indirizzabile è il byte, costituito da 8 bit, cifre del sistema di numerazione binario.
Nel sistema di numerazione binario valgono le stesse regole del sistema di numerazione decimale per la costruzione dei numeri, solo che le cifre disponibili sono due, lo zero e l'uno, e il valore posizionale delle cifre viene calcolato in base alle potenze di 2, invece che alle potenze di 10.
In un byte, il cui valore minimo corrisponde a otto bit 0, e il cui valore massimo corrisponde a otto bit 1, l'intervallo dei numeri rappresentabili è in decimale 0...255.
Nella memoria del calcolatore sono registrati dei bit; la memoria puó essere letta interpretando i bit in binario o in un'altra base di numerazione. Le basi normalmente utilizzate sono la base 10, la base 2, la base 8 e la base 16. Queste ultime due basi vengono utilizzate per comodità dal momento che 2↑3=8 e 2↑4=16, e quindi, raggruppando i bit a 3 a 3 oppure a 4 a 4, si possono facilmente leggere i numeri in queste basi. Il passaggio alla base 10 richiede semplici calcoli.
Il lettore non deve preoccuparsi dei cambiamenti di base tra i numeri, infatti il MONITOR consente di ottenere la rappresentazione di un numero nelle quattro basi citate, basta entrare in MONITOR, scrivere il numero con il prefisso adatto e premere il RETURN, per vedere comparire la rappresentazione completa. I prefissi sono:
- + per i numeri in base 10;
- $ per i numeri in base 16;
- & per i numeri in base 8;
- % per i numeri in base 2.

Con il MONITOR si possono convertire numeri positivi (in valore assoluto); il valore esadecimale massimo consentito è $FFFF.
Inoltre il BASIC 7.0 mette a disposizione due funzioni che consentono la conversione da decimale a esadecimale e viceversa; esse sono la HEX$ e la DEC.

Gli indirizzi riconosciuti sono compresi nell'intervallo 0...65535, cioè sono rappresentati in due byte. Il processore li tratta facendo precedere il byte meno significativo, LOW, a quello più significativo, HIGH. Per esempio l'indirizzo 35478, in esadecimale 8A96, viene scritto in due byte così: 96 8A.

Per quanto riguarda i dati, il mini-assembler disponibile non consente di trattarli come variabili di tipo diverso, come in BASIC; deve pensare l'utente a sistemarli come vuole nei byte e a scrivere routine adatte per elaborarli. Per esempio, l'utente puó scrivere una routine in linguaggio macchina che tratta un numero rappresentato in formato floating point; basta caricare i byte nell'accumulatore del sistema chiamato FAC1 (formato dai byte necessari per memorizzare mantissa e caratteristica del numero) e utilizzare le routine del sistema che lavorano su FAC1.
Il linguaggio macchina puó solo interpretare i dati numerici in due modi diversi:
- in binario;
- in codice BCD.

Abbiamo già visto quale puó essere il contenuto di un byte in binario; il codice BCD utilizza quattro bit per rappresentare una cifra decimale, in conseguenza in un byte possono essere rappresentate due cifre decimali. I codici sono:

bin.	dec.	bin.	dec.	bin.	dec.
0000	0	0001	1	0010	2
0011	3	0100	4	0101	5
0110	6	0111	7	1000	8
1001	9				

i codici da 1010 a 1111 non sono validi.
Il processore dispone di istruzioni per eseguire calcoli con i numeri in entrambe le rappresentazioni.

Il processore oltre ad avere accesso a tutti gli indirizzi di memoria dispone di alcuni registri particolari, il cui utilizzo viene descritto nel capitolo 4.

1.6 DAL PROBLEMA AL PROGRAMMA

Mentre non risulta difficile impadronirsi delle regole necessarie per poter scrivere sequenze di istruzioni BASIC che non contengono errori, la situazione è completamente diversa se si vuole impostare bene un programma non banale. Ci proponiamo di suggerire un metodo di lavoro per passare dal problema al programma.

Il primo passo consiste nella stesura del testo del problema. Due possibili esempi possono essere i seguenti, che indichiamo con i numeri 1 e 2:

- 1) Calcolare le radici di un'equazione di secondo grado.

- 2) Creare e gestire un archivio di indirizzi.

Si tratta di due problemi abbastanza diversi e questo ci permette di impostare bene la discussione. Il problema 1 ha pochi dati di ingresso, solo i tre coefficienti dell'equazione, deve svolgere brevi calcoli, ma puó presentarsi la situazione in cui l'equazione non ammette radici reali, oppure è indeterminata; i risultati sono, se esistono, le due radici reali, oppure le due radici complesse coniugate. Si deve decidere se segnalare soltanto il fatto che esistono radici non reali, o se calcolarle anche in questo caso. Si deve decidere come presentare i risultati.
Il problema 2 puó avere anche molti dati di ingresso, si tratta di conoscere le dimensioni dell'archivio, cioè quanti indirizzi si devono gestire, e i calcoli sono quasi inesistenti. Inoltre il testo è abbastanza generico e non precisa quali utilizzi deve avere l'archivio. Di norma un archivio di indirizzi deve contenere almeno le seguenti informazioni per ogni elemento:
– Cognome e Nome o Ragione Sociale;
– Indirizzo (Via, CAP, Città e Provincia);
ma puó essere richiesto anche il telefono ed eventualmente il recapito telex, o altro. Inoltre a seconda dell'utilizzo dell'archivio si deve stabilire in quale ordine tenere gli elementi. Di norma l'ordinamento è in base al cognome e al nome (o alla ragione sociale), ma puó essere utile tenere raggruppati gli elementi in ordine alfabetico nell'ambito della provincia o altro.
Le operazioni fondamentali da eseguire per creare e gestire un archivio sono le seguenti:
– aggiunta elementi (e quindi anche creazione ex-novo dell'archivio);
– cancellazione elementi;
– modifica elementi;
– visualizzazione o stampa di tutto o di parte dell'archivio.

Per quanto riguarda il problema 1, qualunque sistema calcolatore va bene, le periferiche non devono avere particolari caratteristiche; meglio se il sistema consente di utilizzare una funzione per estrarre la radice quadrata di un numero reale, ma non è indispensabile.
Per il problema 2 invece le periferiche disponibili giocano un ruolo importante; è molto diverso sapere di avere a disposizione un registratore a nastro come memoria di massa, uno o due floppy disk, piuttosto che un hard disk. Vanno inoltre considerate, nel caso si disponga almeno di un floppy disk, le caratteristiche del software disponibile per la gestione dei file su disco. E' infatti molto diverso impostare un archivio con gli elementi in ordine sequenziale e con il solo accesso sequenziale ad essi, che non avere la possibilità di gestire file sequenziali con indice o ad accesso diretto mediante chiave.

Ogni problema si presenta sotto due aspetti:
– *a*) capire il problema nella sua essenza;
– *b*) risolverlo tenendo conto delle risorse del sistema calcolatore di cui si dispone.

Per affrontare il punto a è necessario riflettere sul testo del problema, prendere nota accuratamente dei dati di input e di output, schematizzare le operazioni da effettuare sui dati di input per arrivare ai dati di output, cioè trovare e descrivere gli algoritmi risolutivi. Questo lavoro costituisce l'analisi preliminare del problema; essa deve essere condotta con ordine e deve produrre annotazioni scritte, nelle quali vanno messi in rilievo i punti più delicati delle diverse procedure.
Per condurre a termine l'analisi del problema è necessario affrontare il punto b e quindi devono essere note le risorse del sistema calcolatore di cui si dispone.

Un metodo raccomandato per portare avanti con successo l'analisi di un problema

è quello chiamato TOP-DOWN, cioè dall'alto verso il basso. Questo metodo consiste nel descrivere un problema per livelli, partendo dal testo e scendendo via via alla descrizione particolareggiata di tutte le procedure necessarie. Il vantaggio è che non viene mai persa di vista la visione d'insieme e non si rischia di aver sviluppato molto bene una procedura particolare e di non sapere poi come collegarla alle altre parti.
Il metodo contrario, che parte dal basso e va verso l'alto, denominato BOTTOM-UP, puó rischiare di far lavorare di più, perché alla fine si possono avere tanti mattoni perfetti, che peró non riescono a stare insieme. In realtà, quando si sviluppano programmi complessi, spesso si ricorre a metodi misti, cioè dopo la descrizione dei primi livelli si individuano procedure particolarmente delicate e si progettano in modo da rendersi conto delle difficoltà di collegamento e poi si torna ai livelli superiori.

Il BASIC è un linguaggio che non impone la dichiarazione preliminare di tutte le variabili utilizzate, ció non toglie che si debba preparare l'elenco completo di tutte le variabili, con un commento esplicativo, e tenerlo presente quando si scrive il programma.

Gli algoritmi utilizzati devono essere descritti. Tutta la procedura di risoluzione di un problema deve essere descritta. Questo si puó ottenere in diversi modi:
– con una descrizione per punti, identificabili per mezzo di numeri o di sigle alfanumeriche, in modo che sia possibile fare dei riferimenti;
– con una descrizione grafica mediante schemi, chiamati diagrammi a blocchi, costituiti da simboli grafici diversi, ognuno con un particolare significato, all'interno dei quali brevi frasi contribuiscono alla comprensione. I simboli grafici sono collegati da linee orientate che mostrano immediatamente i collegamenti tra le varie procedure.
– con una descrizione mediante uno pseudo linguaggio di progetto basato su regole meno restrittive di quelle di un vero linguaggio di programmazione, ma significative per la comprensione del problema e per il successivo passaggio alla codifica del programma.

Nel seguito di questo libro noi riportiamo parecchi programmi come esempio, in BASIC e in linguaggio macchina; in ogni caso diamo una descrizione delle variabili utilizzate e delle procedure. Per ovvie ragioni i programmi non sono lunghi e complessi, per cui non ci preoccupiamo di stendere una documentazione completa, che peró raccomandiamo al lettore. Infatti è estremamente facile dimenticare perché si sono fatte alcune cose o si sono scelte certe procedure, anche se la stesura di un programma è costata molto lavoro.

La documentazione di un programma consiste nella descrizione dei dati di input e di output, delle tabelle di dati utilizzate, delle aree di lavoro e degli algoritmi. Inoltre è necessario preparare un manuale operativo, cioè la spiegazione delle modalità di utilizzo, comprendente il significato dei messaggi e le risposte possibili. Attualmente si tende a corredare i programmi con brevi spiegazioni di aiuto che compaiono su una parte del video mentre si lavora; questo è molto utile ma non elimina la necessità del manuale operativo.

CAPITOLO 2

PROGRAMMARE IN BASIC 7.0

2.1 INGRESSO E USCITA DEI DATI CON TASTIERA E VIDEO

La tastiera e il video sono le periferiche principali del COMMODORE 128. Il file di input corrispondente alla tastiera non ha bisogno di essere aperto o chiuso, e così pure il file di output corrispondente al video. Le istruzioni di input e output scritte senza il suffisso '#n' agiscono a livello tastiera e video. Risulta inoltre possibile trattare anche la tastiera e il video come due periferiche qualsiasi, assegnando loro un numero logico di file, utilizzando le istruzioni OPEN e CLOSE e le istruzioni di input e output con il suffisso '#n'.

La tastiera ha un buffer che puó contenere 10 caratteri, posizionato nelle locazioni di memoria da 842 a 851. Il calcolatore scandisce la tastiera 60 volte al secondo per rilevare la pressione dei tasti. La pressione di un tasto provoca la memorizzazione di un codice particolare nella locazione 212; ad ogni tasto corrisponde un codice diverso, che dipende dalla posizione del tasto. Inoltre nella locazione 211 viene memorizzato un numero (una potenza di 2) per segnalare la pressione di uno dei tasti di controllo SHIFT, COMMODORE, CONTROL e ALT; i valori sono:
- 0 se non è stato premuto alcun tasto di controllo;
- 1 se è stato premuto SHIFT;
- 2 se è stato premuto COMMODORE;
- 4 se è stato premuto CONTROL;
- 8 se è stato premuto ALT.

In base ai codici presenti nelle locazioni 211 e 212 il sistema memorizza nella posizione del buffer puntata dalla locazione 208 il codice ASCII del carattere. Quando la locazione 208 contiene zero il buffer è vuoto. Il buffer viene svuotato durante l'esecuzione delle operazioni di ingresso dati, ma se queste non agiscono nei tempi previsti possono andare persi dei caratteri.

Le routine utilizzate per l'esecuzione delle istruzioni di input e output tramite le periferiche principali producono gli effetti che elenchiamo.
- GET, legge un carattere in ogni variabile della lista senza attendere la pressione di un tasto. E' consigliabile utilizzare nella lista solo variabili di tipo stringa per evitare il messaggio d'errore 'TYPE MISMATCH ...' che si verifica quando si tenta di ricevere caratteri non numerici in variabili di tipo numerico. Il programma prosegue anche se il buffer della tastiera è vuoto e non si premono tasti. I caratteri letti con GET non vengono visualizzati sul video.
- GETKEY, legge un carattere in ogni variabile della lista per ogni tasto premuto. Anche in questo caso è consigliabile utilizzare nella lista variabili di tipo stringa. Il programma non prosegue fino a quando non si premono tutti i tasti attesi. I caratteri letti con GETKEY non vengono visualizzati sul video.
- INPUT, legge tanti dati quante sono le variabili della lista. Consente di premettere un messaggio alla lista delle variabili, che viene di norma utilizzato per richiedere i dati. Le variabili della lista sono separate da virgole. I dati di input vengono

separati da virgole, che sono riconosciute come caratteri separatori; questo puó creare dei problemi se si desidera includere delle virgole nelle stringhe di caratteri, ma si puó superare la difficoltà aprendo le virgolette prima dei dati alfanumerici e chiudendole dopo. Inoltre si deve tener presente che il sistema elimina gli spazi eventualmente presenti prima e dopo i dati; in conseguenza se si desidera includere degli spazi nei dati alfanumerici questi devono essere scritti tra virgolette. Altra cosa da tener presente è che se alla richiesta di un dato si risponde solo con il tasto RETURN la variabile interessata mantiene il suo precedente valore. I caratteri letti con INPUT vengono visualizzati sul video quando vengono prelevati dal buffer della tastiera.
– PRINT, visualizza le variabili della lista con il loro formato, riconosce i caratteri di controllo che agiscono sulla posizione del cursore e tiene conto della punteggiatura che separa le variabili. I numeri interi vengono visualizzati con il loro numero di caratteri, preceduti da uno spazio o dal segno meno e seguiti da uno spazio. I numeri non interi vengono visualizzati in formato *virgola fissa* (cioè con il punto decimale) fino a 9 cifre, escludendo un piccolo intervallo intorno allo zero, e in formato esponenziale negli altri casi; anche essi sono preceduti da uno spazio o dal segno meno e seguiti da uno spazio. Nella figura 2.1 è schematizzato il comportamento in stampa dei numeri non interi.

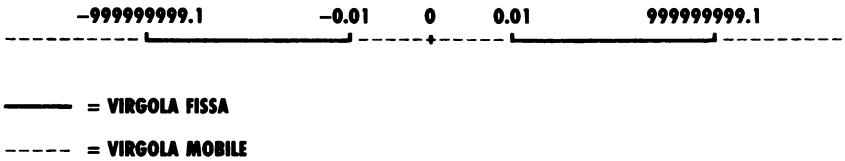


Fig. 2.1 Intervalli per la stampa dei numeri non interi

Le stringhe di caratteri vengono visualizzate come sono.
La funzione TAB fa avanzare il cursore nella posizione indicata contando dall'inizio della linea con eventuale passaggio alla linea successiva, ma non agisce se tale posizione è già stata superata.
La funzione SPC provoca il salto del numero di posizioni indicato partendo dalla posizione attuale del cursore, anche con eventuale passaggio alla linea successiva.
La virgola separatrice tra le variabili della lista fa continuare la visualizzazione dalla prossima zona di stampa; ogni zona di stampa è di 10 caratteri.
Il punto e virgola separatore provoca la stampa dei dati uno dopo l'altro nel loro formato.
Se la lista di stampa termina senza punteggiatura il cursore alla conclusione dell'operazione si porta all'inizio della linea successiva; se invece la lista termina con virgola o punto e virgola la prossima stampa continua sulla stessa riga, rispettando le regole della punteggiatura.
Per provocare il posizionamento in punti determinati dello schermo si possono utilizzare stringhe di caratteri di controllo tra virgolette.
Utilizzando la forma PRINT USING si puó influire sul modo di stampare i dati per mezzo della maschera di formato, che deve essere scritta tra virgolette e separata dal punto e virgola dalla lista di stampa.

La parola chiave PRINT puó essere sostituita dal punto interrogativo con lo stesso significato.

Il video ha come numero di periferica il 3 e con tale numero puó essere utilizzato sia come file di input che di output; la tastiera ha come numero di periferica 0. Aprendo il video e la tastiera come file si devono utilizzare le istruzioni di input e output seguite dal suffisso '#n', senza spazi intermedi; in questo caso non si puó scrivere il punto interrogativo al posto della parola PRINT.

Il programma ES1 puó essere utile per eseguire alcune prove di ingresso dati; esso chiede e visualizza un numero intero, un numero non intero e una stringa. Si possono constatare i seguenti fatti:
– rispondendo con un numero intero non compreso nell'intervallo -32768/+32767 si interrompe il programma con il messaggio 'ILLEGAL QUANTITY ...';
– rispondendo con un numero non intero che in formato esponenziale ha un esponente non compreso nell'intervallo -38/+37 il programma si interrompe con il messaggio 'OVERFLOW ERROR ...';
– rispondendo invece che con un numero con una stringa alfabetica il programma segnala 'REDO FROM START' e chiede nuovamente il dato;
– rispondendo con una stringa con più di 255 caratteri il programma si interrompe con il messaggio 'STRING TOO LONG ...'.

```
1 REM ES1
3 INPUT "SCRIVI UN NUMERO INTERO: ";N%
5 PRINT "HAI SCRITTO: ";N%
7 INPUT "SCRIVI UN NUMERO NON INTERO: ";N
9 PRINT "HAI SCRITTO: ";N
11 INPUT "SCRIVI UNA FRASE: ";A$
13 PRINT "HAI SCRITTO: ";A$
```

Il programma ES2 puó essere utilizzato per vedere quali codici corrispondono alla pressione dei tasti anche utilizzando i tasti di controllo SHIFT, COMMODORE, CONTROL e ALT.

```
1 REM ES2
3 PRINT "PREMI UN TASTO"
5 GETKEY A$
7 PRINT "CODICE TASTO = ";PEEK(212)
9 PRINT "211 CONTIENE: ";PEEK(211)
11 GOTO 3
```

Viene chiesto di premere un tasto, che viene letto con GETKEY, ma invece di visualizzare la variabile A$ utilizzata, vengono letti con la funzione PEEK e mostrati i contenuti dei due byte 212 (codice del tasto) e 211 (tasto di controllo). Si possono eseguire diverse prove e scoprire quali sono i codici dei tasti. Per interrompere il programma premere contemporaneamente RUN/STOP e RESTORE.

Il programma ES3 mostra come si puó leggere dal buffer della tastiera senza premere tasti.

```
1 REM ES3
3 GOSUB 37:REM RIEMPIE IL BUFFER
5 B$="": REM B$ UGUALE STRINGA NULLA
7 FOR K=1 TO 10: REM LEGGE DAL BUFFER
9 GETKEY A$:B$=B$+A$: REM CON GETKEY
11 NEXT K
13 PRINT B$: REM STAMPA CARATTERI
15 GOSUB 37: REM RIEMPIE IL BUFFER
17 INPUT C$: REM LEGGE CON INPUT DAL BUFFER
19 PRINT C$: REM STAMPA CARATTERI
21 GOSUB 37: REM RIEMPIE IL BUFFER
23 OPEN 1,0: REM APRE LA TASTIERA COME FILE
25 INPUT#1,D$: REM LEGGE CON INPUT#1 DAL BUFFER
27 PRINT
29 PRINT D$: REM STAMPA CARATTERI
31 CLOSE 1
33 PRINT LEN(B$),LEN(C$),LEN(D$)
35 STOP
37 FOR K=842 TO 850: REM SOTTOPROGRAMMA
39 POKE K,K-777: REM PONE NEL BUFFER I
41 NEXT K: REM CODICI ASCII DELLE LETTERE
43 POKE 851,13: REM DA A AD I E RETURN
45 POKE 208,10: REM 10 CARATTERI NEL BUFFER
47 RETURN
```

Il sottoprogramma delle linee 37/47 riempie il buffer della tastiera con i codici ASCII da 65 a 73 e il codice 13 finale (RETURN), e pone 10 nel byte 208, puntatore al buffer della tastiera, a significare buffer pieno. Il programma, dopo aver richiamato il sottoprogramma e aver posto la variabile B$ a stringa nulla, esegue 10 istruzioni GETKEY A$, aggiungendo ogni carattere in B$, cioè svuota il buffer della tastiera, e poi visualizza B$. Il programma prosegue richiamando nuovamente il sottoprogramma, leggendo dal buffer con l'istruzione INPUT C$ e visualizzando C$. La terza volta, dopo aver riempito nuovamente il buffer, la tastiera viene aperta come file e i dati vengono letti con INPUT#1; si noti l'istruzione PRINT alla linea 27, necessaria per non sovrapporre parzialmente la stampa con i caratteri visualizzati durante lo svuotamento del buffer, dato che in questo caso il cursore non va a capo e resta sulla lettera I. Infatti il codice del RETURN viene interpretato solo come separatore di campo e non manda a capo come nel caso precedente. Alla linea 33 vengono visualizzate le lunghezze delle tre stringhe lette. Il risultato sul video è il seguente:

```
ABCDEFGHI

? ABCDEFGHI
ABCDEFGHI
ABCDEFGHI
ABCDEFGHI
  10  .   9        9
```

Si noti che dopo la stampa di B$ si ha una riga bianca per effetto del codice 13 presente nella stringa, mentre la PRINT della linea 27 fa solo andare a capo; inoltre dalle lunghezze delle tre stringhe, visualizzate sull'ultima riga, si vede che la prima

stringa ha un carattere in più, proprio il codice 13. Come si puó vedere dai risultati i caratteri vengono visualizzati anche durante il prelevamento dei dati dal buffer con le istruzioni INPUT e INPUT # 1, e nel caso di INPUT # 1 non viene visualizzato il punto interrogativo.

Abbiamo visto che durante l'ingresso dei dati il sistema puó segnalare alcuni tipi di errori; inoltre se la lista dei dati da leggere comprende più di una variabile e si risponde con meno dati il sistema va a capo e visualizza due punti interrogativi per avvisare che l'input non è terminato. Anche i messaggi che non provocano interruzione del programma possono dare fastidio perché rovinano il quadro video. In conseguenza proponiamo alcuni esempi di programmi che riguardano il controllo dei dati di input e la preparazione di quadri video per la presentazione di menu di scelta e l'immissione dei dati.

Il programma ES4 legge un dato numerico con eventuale punto decimale, dopo averne chiesto la lunghezza massima, come stringa di caratteri con l'istruzione GETKEY, e controlla che sia di tipo numerico, cioè, che possa essere presente il segno solo in prima posizione, che il punto decimale compaia al massimo una volta e i caratteri siano numerici. Il programma accetta il carattere DEL e cancella un carattere; inoltre se il carattere precedente al DEL è stato riconosciuto errato lo cancella e annulla l'errore consentendo di procedere. Esso utilizza le seguenti variabili:

- E, indicatore di errore per dato che non va accettato;
- P, indicatore di errore per un solo punto decimale;
- S, indicatore per consentire di annullare l'errore se seguito dal carattere DEL (CHR$(20));
- L, lunghezza del dato;
- N$, dato finale;
- A$, lettura di un carattere;
- B$, carattere precedente;
- K, controllo dei cicli.

Il programma dopo aver chiesto il numero massimo di caratteri da leggere, L, e aver visualizzato il messaggio di richiesta del dato (linea 15), utilizza il sottoprogramma delle linee 21/67 per leggere e controllare il dato. Il sottoprogramma ritorna l'indicatore E a zero se il dato puó essere accettato.

```
1 REM ES4
3 REM CONTROLLO DATO NUMERICO
5 REM LETTO CON GET COME STRINGA
7 REM NOTA LA LUNGHEZZA MASSIMA DEL DATO
9 INPUT "QUANTI CARATTERI: ";L
11 IF L=0 THEN STOP
13 IF L<0 THENPRINT "?":GOTO 9
15 PRINT "SCRIVI DATO":GOSUB 21
17 IF E=0 THEN PRINT "ACCETTATO: " N$:GOTO 9
19 PRINT "NON VALIDO":GOTO 9
21 REM SOTTOPROGRAMMA LETTURA E CONTROLLO
23 REM INIZIALIZZA VARIABILI
25 N$="":E=0:P=0:S=0:B$=""
27 PRINT "DATO: ";
29 FOR K=1 TO L
```

```
31 GETKEY A$
33 REM ACCETTA DATO CON MENO CARATTERI
35 IF A$=CHR$(13) THEN 65
37 REM CONTROLLO CARATTERE LETTO
39 IF A$>"/" AND A$<":" THEN 59
41 IF K=1 AND (A$="+" OR A$="-") THEN 59
43 IF P=0 AND A$="." THEN P=1:GOTO 59
45 IF S=0 AND A$=CHR$(20) THEN 53
47 IF S=1 AND A$=CHR$(20) THEN S=0:GOTO 31
49 IF S=1 THEN 57
51 S=1:GOTO 31
53 IF B$="." THEN P=0
55 PRINT "▮ ▮";:GOTO 67
57 E=1:GOTO 65
59 B$=A$:N$=N$+A$:PRINT A$;
61 REM PULISCE BUFFER PRIMA DI USCIRE
63 NEXT K:PRINT:POKE 208,0:RETURN
65 K=L:NEXT K:PRINT:RETURN
67 N$=LEFT$(N$,LEN(N$)-1):GOTO 31
```

Il sottoprogramma accetta anche un dato con meno caratteri di L. Si noti che ogni carattere letto, se valido, viene aggiunto alla stringa N$, che alla fine contiene il dato completo, e conservato in B$ per eventuali controlli derivanti dal carattere successivo. Si noti come alla linea 67 N$ viene privato dell'ultimo carattere da cancellare, e come viene cancellato un carattere dal video alla linea 55.

Il programma ES5, invece, dopo aver chiesto la lunghezza massima L del dato numerico da leggere, dello stesso tipo di quello trattato dal programma precedente, lo legge come stringa con l'istruzione INPUT e controlla i caratteri letti, utilizzando il sottoprogramma delle linee 27/53. Sono consentiti il segno come primo carattere, un solo punto decimale e i caratteri numerici. Alla fine se il dato è più corto di L vengono aggiunti spazi di riempimento a destra. Variabili utilizzate:
- L, lunghezza del dato;
- K, controllo dei cicli;
- S$, contiene L spazi;
- E, indicatore di errore;
- Z$, lettura di un carattere;
- P, controllo presenza di un solo punto decimale;
- A$, dato;
- B$, un carattere del dato.

```
1 REM ES5
3 REM CONTROLLO DATO NUMERICO
5 REM LETTO CON INPUT COME STRINGA
7 REM NOTA LA LUNGHEZZA MASSIMA DEL DATO
9 INPUT "▯QUANTI CARATTERI: ";L
11 IF L<=0 THEN PRINT "▯":GOTO 9
13 S$="":FOR K=1 TO L:S$=S$+" ":NEXT K
15 GOSUB 27
```

```
17 IF E<>0 THEN 21
19 PRINT "■■■■ACCETTATO: " A$:GOTO 23
21 PRINT "■■■■NON VALIDO"
23 PRINT "PREMI UN TASTO PER CONTINUARE"
25 GETKEY Z$:GOTO 9
27 REM SOTTOPROGRAMMA LETTURA
29 PRINT "■■■■■■■■■■■■■■■■■■■■■■■■"
31 P=0:E=0:A$="":INPUT "DATO: ";A$
33 REM AGGIUSTA LA LUNGHEZZA SE MAGGIORE
35 IF LEN(A$)>L THEN A$=LEFT$(A$,L)
37 REM CONTROLLO CARATTERI LETTI
39 FOR K=1 TO LEN(A$):B$=MID$(A$,K,1)
41 IF K<>1 THEN 45
43 IF B$="+" OR B$="-" THEN 53
45 IF B$>"/" AND B$<":" THEN 53
47 IF B$="." AND P=0 THEN P=1:GOTO 53
49 E=E+1:K=L:NEXT K:RETURN
51 REM AGGIUNGE SPAZI SE NECESSARIO
53 NEXT K:A$=LEFT$(A$+S$,L):RETURN
```

Il dato di input viene letto visualizzandolo nella parte bassa del video; per spostarsi sul video viene utilizzata una stringa di caratteri di controllo formata dal carattere iniziale HOME e da una sequenza di caratteri CRSR/DOWN. Si noti come alle linee 23 e 25 viene chiesto di premere un tasto per continuare. L'indicatore P serve per controllare la presenza di un solo punto decimale e l'indicatore E per fornire al programma principale informazioni sulla validità del dato (E=0).

Il programma ES6 mostra come si possa ricevere la data sovrapponendola ad una maschera di richiesta e come si possa visualizzare un menu che accetta la scelta solo con la pressione di un tasto (istruzione GETKEY), ma controllandola. Nella presentazione del menu si fa uso del campo inverso per evidenziare alcune parti.
Il programma utilizza diversi sottoprogrammi: per la richiesta della data (linee 49/63), per la richiesta di conferma (linee 67/75) e per la richiesta della pressione di un tasto per proseguire (linee 77/87). Vengono utilizzate le stringhe di caratteri di controllo per spostarsi sul video, sia orizzontalmente che verticalmente. Le variabili utilizzate sono:
- CH$, carattere CLEAR/HOME;
- SZ$, 8 spazi;
- G$, M$, A$, giorno, mese, anno;
- R$, risposte;
- T$, scelta opzione;
- A, valori numerici;
- DT$, data globale;
- CD$, caratteri controllo posizione video;
- M$, messaggio.

```
1 REM ES6
3 CH$="■":SZ$="        "
5 CD$="■■■■■■■■■■■■■■■■■■■■■■■■"
```

```
7 REM RICHIEDE LA DATA E LA STAMPA
9 GOSUB 49:PRINT "□□□" SPC(8) "DATA ODIERNA: ";
11 PRINT G$;"/";M$;"/";A$
13 GOSUB 65
15 IF R$="N" THEN 9
17 REM PRESENTAZIONE MENU SCELTA
19 PRINT "□";:COLOR 4,7:COLOR 0,1
21 PRINT CH$;SZ$ "□□□   ■PROGRAMMA XXXXXX■"
23 PRINT
25 PRINT:PRINT SZ$" ■1■ - INIZIALIZZAZIONE"
27 PRINT:PRINT SZ$" ■2■ - AGGIORNAMENTO"
29 PRINT:PRINT SZ$" ■3■ - PROCEDURA CALCOLO"
31 PRINT:PRINT SZ$" ■4■ - STAMPA RISULTATI"
33 PRINT:PRINT SZ$" ■0■ - FINE"
35 PRINT:PRINT:PRINT SZ$"      ■OPZIONE: ";
37 REM SCELTA OPERAZIONE
39 DO UNTIL T$>="0" AND T$<="4":GETKEY T$:LOOP
41 PRINT "■"+T$:GOSUB 79
43 A=VAL(T$):IF A=0 THEN 99
45 ON A GOTO 91,93,95,97
47 STOP
49 REM RICHIESTA DATA
51 COLOR 4,5:COLOR 0,2
53 PRINT "□□□□□□":OPEN 1,0
55 PRINT SZ$"   DATA  GG,MM,AA■■■■■■■■■";
57 INPUT#1,G$:PRINT "■■■■";
59 INPUT#1,M$:PRINT "■";
61 INPUT#1,A$
63 DT$=G$+M$+A$:CLOSE 1:RETURN
65 REM RICHIESTA CONFERMA
67 OPEN 1,0
69 PRINT LEFT$(CD$,21);
71 PRINT SZ$"   CONFERMI ?  S■";:INPUT#1,R$
73 IF R$<>"S" AND R$<>"N" THEN 67
75 CLOSE 1:RETURN
77 REM RICHIESTA TASTO PER PROSEGUIRE
79 PRINT CD$;
81 M$="■PREMI UN TASTO PER PROSEGUIRE■"
83 PRINT "    "M$
85 GETKEY T$
87 PRINT CH$:RETURN
89 REM SEGNALA LA SCELTA OPERATA
91 STOP
93 STOP
95 STOP
97 STOP
99 STOP
```

Nel programma viene utilizzata l'istruzione COLOR per modificare i colori del bordo e dello sfondo. La tastiera viene aperta come file per evitare di vedere il punto interrogativo con l'istruzione INPUT. Si noti come si procede nelle linee da 49 a 63 per ricevere le risposte GG, MM, AA, sovrapponendole alla maschera GG,MM,AA. Dopo ogni risposta si deve premere RETURN, ma utilizzando l'istruzione INPUT#1 il cursore non va a capo, poi la stringa di caratteri CRSR/LEFT fa raggiungere la posizione voluta. La data peró non viene controllata numericamente e quindi si deve fare attenzione nella risposta.
Alla linea 71 si noti come è stata scritta la stringa che richiede la conferma, facendo lampeggiare il cursore sulla S.
Il controllo sulla risposta al menu di scelta è ottenuto utilizzando l'istruzione DO UNTIL ... LOOP, che continua a leggere un carattere fino a quando questo risulta compreso tra 0 e 4, estremi inclusi.
Il programma segnala la scelta operata sul menu con una serie di STOP con numero di linea diverso.
Il programma ES7 utilizza ampiamente la tecnica dei sottoprogrammi; esso mostra come si possa dimensionare, lavorando sul video, una struttura di dati formata da 15 campi con diciture e lunghezze diverse e come si possano ricevere e controllare detti dati. La funzione di dimensionamento della struttura di dati è svolta dal sottoprogramma situato nelle linee 125/205. Questo sottoprogramma, o uno analogo, puó essere utilizzato per l'inizializzazione di un file su disco; esso dovrebbe essere completato con una parte che genera un file sequenziale contenente le informazioni necessarie per gestire un file che ha la struttura del record da esso definita. Il sottoprogramma verifica che l'insieme dei dati non superi 237 caratteri e che la lunghezza di ogni campo non superi 30 caratteri. Si noti come si ottiene di poter modificare le diciture e le lunghezze nelle linee 149/157: premendo RETURN il campo resta invariato, scrivendo sopra il vecchio valore si puó modificare il campo. Alla fine se i caratteri sono troppi viene chiesto di modificare la struttura.
Dopo la definizione della struttura dei dati essi vengono richiesti elencandoli sul video, ne viene chiesta conferma ed è possibile modificare i campi. Le variabili utilizzate sono:
- L, numero campi;
- D$(L), descrizione campi;
- L(L), lunghezze campi;
- R$(L), campi;
- M$(11), MS$, MT$, messaggi;
- M(11), lunghezze campi diciture;
- CH$, CD$, caratteri controllo video;
- SZ$, SP$, spazi;
- I, IK, J, controllo cicli;
- A$, R$, lettura carattere;
- F, switch di controllo;
- R, numero campo;
- IL, lunghezza campo;
- YX, controllo lunghezze.

```
1 REM ES7
3 REM DEFINIZIONE COSTANTI E VARIABILI
5 L=15:DIM D$(L),L(L),M$(11),M(11),R$(L)
7 CH$="J":CD$="SSSSSSSSSSSSSSSSSSSSSSSSSSSSSS"
9 SZ$="                              "
```

```
11 PRINT "▐";:FOR I=1 TO 30:SP$=SP$+" ":NEXT I
13 DATA 8,6,9,0,5,5,6,8,0,4,9
15 FOR K=1 TO 11:READ M(K):NEXT K
17 M$(1)="▐▐▐▐SCELTA NOMI DEI 15 CAMPI"
19 M$(2)="CONFERMA IL NOME SE VA BENE"
21 M$(3)="ALTRIMENTI RISCRIVILO."
23 M$(4)=""
25 M$(5)="OSSERVA LE LUNGHEZZE DEI CAMPI"
27 M$(6)="PUOI CAMBIARLE. LA SOMMA DELLE"
29 M$(7)="LUNGHEZZE DEI CAMPI NON PUO'"
31 M$(8)="SUPERARE 237 CARATTERI,"
33 M$(9)=""
35 M$(10)="LA LUNGHEZZA DI UN CAMPO NON PUO'"
37 M$(11)="SUPERARE 30 CARATTERI."
39 MS$="       ▐PREMI UN TASTO PER PROSEGUIRE▄"
41 MT$="         QUALE CAMPO      ▌▌▌▌"
43 REM RICHIESTA E CONTROLLO DATI
45 PRINT CH$;:GOSUB 125
47 GOSUB 65:STOP
49 PRINT CD$;MS$
51 REM CHIEDE UN TASTO PER CONTINUARE
53 GETKEY A$:PRINT CH$:RETURN
55 REM CHIEDE CONFERMA
57 CLOSE 1:OPEN 1,0:PRINT LEFT$(CD$,21);
59 PRINT SZ$ "   CONFERMI    S▌";
61 INPUT#1,R$:IF R$<>"S" AND R$<>"N" THEN 55
63 CLOSE 1:RETURN
65 REM RICHIESTA E CONTROLLO DATI
67 CLOSE 1:OPEN 1,0
69 PRINT "▐  ▐ INGRESSO DATI▐"
71 F=0:PRINT " 1";TAB(4);D$(1)+":←▌";
73 INPUT#1,R$(1):PRINT
75 REM SE RISPOSTA SOLO RETURN ESCE
77 IF LEFT$(R$(1),1)="←" THEN F=1:RETURN
79 REM LEGGE I DATI
81 FOR I=2 TO L:PRINT I;TAB(4);D$(I)":←▌";
83 INPUT#1,R$(I):PRINT:NEXT I:GOSUB 99
85 GOSUB 55:IF R$="S" THEN CLOSE 1:RETURN
87 CLOSE 1:OPEN 1,0
89 REM CHIEDE CORREZIONI
91 PRINT LEFT$(CD$,21);MT$;:INPUT R
93 IF R<1 OR R>L THEN 91
95 IF R>L OR R<1 GOTO 91
97 GOSUB 107:GOTO 85
99 REM AGGIUSTA LUNGHEZZA DATI
101 FOR I=1 TO L
103 R$(I)=LEFT$(R$(I)+SP$,L(I))
105 NEXT I:RETURN
107 REM STAMPA I DATI
```

```
109 PRINT "▒▒";:FOR I=1 TO R-1
111 PRINT I;TAB(4);D$(I);":";R$(I):NEXT I
113 PRINT R;TAB(4);D$(R);":←▒";:INPUT#1,R$(R)
115 PRINT:R$(R)=LEFT$(R$(R)+SP$,L(R))
117 IF R>=L THEN 123
119 FOR I=R+1 TO L
121 PRINT I;TAB(4);D$(I);":";R$(I):NEXT I
123 RETURN
125 REM PREPARA DICITURE E LUNGHEZZE DATI
127 DW$="S":D$(1)="COGNOME":D$(2)="NOME"
129 D$(3)="INDIRIZZO":D$(4)="TELEFONO"
131 D$(5)="C.A.P.":D$(6)="CITTA'"
133 FOR IK=7 TO L:D$(IK)="C"+CHR$(42+IK):NEXT IK
135 DATA 20,15,30,15,8,15,22,22,15,15
137 DATA 15,15,15,10,5
139 FOR IK=1 TO L:READ L(IK):NEXTIK
141 CLOSE 1:OPEN 1,0:PRINTCH$;
143 FOR K=1 TO 11
145 PRINT SPC(M(K)) "▒" M$(K) "▒":NEXT K
147 GOSUB 49
149 PRINT CH$ "▒CONFERMA STRUTTURA DATI▒"
151 FOR J=1 TO L
153 PRINT "L=";L(J);TAB(10);D$(J):NEXT J
155 PRINT:PRINT "SE VA BENE PREMI RETURN"
157 PRINT "SE NO RISCRIVI PER SOSTITUIRE"
159 REM CHIEDE CONFERMA PER I SINGOLI CAMPI
161 FOR J=1 TO L
163 PRINT LEFT$(CD$,J+1);TAB(10);"▒";D$(J);"▒";
165 PRINT CD$;SP$;CD$;:T$="":INPUT#1,T$
167 IF T$="" THEN 171
169 D$(J)=T$
171 PRINT LEFT$(CD$,J+1);TAB(8);SP$;
173 PRINT LEFT$(CD$,J+1);TAB(10);D$(J)
175 PRINT LEFT$(CD$,J+1);TAB(2);"▒";L(J);"▒";
177 PRINT CD$;SP$;CD$;:IL=0:INPUT#1,IL
179 IF IL=0 THEN 187
181 IF IL>30 THEN 175
183 L(J)=IL
185 PRINT LEFT$(CD$,J+1);TAB(4);"  ";
187 PRINT LEFT$(CD$,J+1);TAB(2);L(J):NEXT J
189 REM CONTROLLA OCCUPAZIONE TOTALE DEI DATI
191 YX=0:FOR J=1 TO L:YX=YX+L(J):NEXT J
193 IF YX<=237 THEN 201
195 PRINT LEFT$(CD$,20);
197 PRINT "TROPPI CARATTERI";YX
199 GOSUB 49:GOTO 149
201 GOSUB 55
203 IF R$="N" THEN 141
205 CLOSE 1:RETURN
```

Il programma è stato preparato utilizzando dei vettori per le diciture dei campi, le lunghezze e i dati stessi; in conseguenza esso risulta facilmente adattabile ad altre situazioni. Le richieste di dati vengono fatte aprendo la tastiera come file, così si evita la comparsa del punto interrogativo. Anche in questo caso si utilizzano le stringhe di caratteri di controllo per spostarsi sul video. I campi sono tutti di tipo stringa e non vengono fatti controlli sui dati. E' possibile aggiungere sottoprogrammi di controllo dei dati del tipo di quelli mostrati negli esempi precedenti. Abbiamo scritto i messaggi come costanti per evitare istruzioni troppo lunghe.

Il programma ES8 mostra invece come si possano ricevere dati dopo aver preparato un quadro video contenente apposite caselle per ogni dato. In questo caso abbiamo utilizzato un solo sottoprogramma, quello di lettura, nelle linee 105/117. Esso utilizza l'istruzione GETKEY e legge per ogni dato solo il numero massimo di caratteri consentiti o meno; se si scrivono tutti i caratteri che stanno nella casella non si deve premere RETURN, mentre lo si deve premere se il campo resta incompleto. Viene azzerato il puntatore al buffer della tastiera (byte 208) dopo ogni lettura di dato per evitare di sporcare i campi successivi. Variabili utilizzate:
- V$, caratteri controllo video;
- SP$, spazi in campo inverso;
- D$(12), diciture campi;
- R$(12), dati campi;
- CR(12), posizioni righe;
- CC(12), posizioni colonne;
- L(12), lunghezze campi;
- T$, messaggio;
- K, I, controllo cicli;
- R$, risposte;

```
1 REM ES8
3 REM DEFINIZIONE COSTANTI
5 V$="■■■
```

```
45 REM BORDO NERO, SFONDO BLU
47 COLOR 4,1:COLOR 0,7
49 REM PREPARAZIONE QUADRO VIDEO
51 PRINT "▯" TAB(12) T$
53 FOR K=1 TO 12
55 PRINT LEFT$(V$,CR(K)+1) TAB(CC(K));D$(K)
57 PRINT LEFT$(V$,CR(K)+2);
59 PRINT TAB(CC(K)) LEFT$(SP$,L(K)+1)
61 NEXT K
63 REM RICHIESTA DATI
65 FOR K=1 TO 12
67 PRINT LEFT$(V$,CR(K)+2) TAB(CC(K));"▯ ▬";
69 GOSUB 107:FOR I=1 TO 100:NEXT I
71 NEXT K
73 POKE 208,0:REM RICHIESTA CONFERMA
75 PRINT V$"                                ";:R$=""
77 PRINT V$"          CONFERMI (S/N)";:INPUT R$
79 IF R$<>"S" AND R$<>"N" THEN 75
81 IF R$="S" THEN 97
83 REM MODIFICA CAMPI
85 POKE 208,0
87 PRINT V$"                                ";:R$=""
89 PRINT V$"           QUALE CAMPO:";:INPUT R$
91 K=VAL(R$):IF K<1 OR K>12 THEN 85
93 PRINT LEFT$(V$,CR(K)+2) TAB(CC(K));"▯ ▬";
95 GOSUB 107:GOTO 73
97 REM STAMPA DATI LETTI
99 PRINT "▯        DATI LETTI"
101 FOR K=1 TO 12:PRINT D$(K);": ";R$(K):NEXT K
103 STOP
105 REM SOTTOPROGRAMMA LETTURA
107 POKE 208,0:R$(K)=""
109 FOR I=1 TO L(K):GETKEY A$
111 IF A$=CHR$(13) THEN I=L(K):GOTO 115
113 PRINT A$;:R$(K)=R$(K)+A$
115 NEXT I
117 POKE 208,0:FOR I=1 TO 100:NEXT I:RETURN
```

Il programma inizialmente prepara il quadro video visualizzando le diciture e le caselle per i dati. Quando il quadro è completo si possono scrivere i dati, ma non più lunghi delle dimensioni visualizzate. Il cursore non si vede, ma compare mentre si scrivono i dati. Alla fine viene chiesta conferma dei dati e se si risponde con N viene chiesto il numero del campo da modificare; in questo caso il cursore si posiziona sulla casella richiesta e si deve riscrivere il dato. Alla fine viene visualizzata la lista dei dati. Nel sottoprogramma di lettura è stato introdotto un ciclo FOR fittizio per allungare i tempi e così pure alla linea 69.
Anche in questo caso non abbiamo aggiunto sottoprogrammi per controllare i dati dei singoli campi. Come sempre utilizziamo stringhe di caratteri di controllo per spostarci sul video.

Abbiamo preparato il programma ES9 per provare tutte le possibili combinazioni di colore per il bordo, lo sfondo e i caratteri, lavorando in modo testo. Il programma è formato da tre cicli FOR concatenati; il ciclo più interno modifica il colore dei caratteri, quello intermedio il colore dello sfondo e il più esterno quello del bordo. Per passare da una configurazione all'altra si deve premere un tasto; in conseguenza il programma risulta un po' lento, ma si ha la possibilità di annotare i numeri dei colori utilizzati per le combinazioni preferite. Variabili utilizzate:
- K, J, I, controllo cicli;
- A$, lettura di un carattere.

```
1 REM ES9
3 FOR K=1 TO 16:REM COLORE BORDO
5 COLOR 4,K
7 FOR J=16 TO 1 STEP-1 :REM COLORE SFONDO
9 COLOR 0,J
11 FOR I=1 TO 16:REM COLORE CARATTERI
13 COLOR 5,I
15 PRINT "▯▯▯▯▯▯▯▯▯COLORE BORDO: ";K
17 PRINT "▮▮▮▮COLORE SFONDO: ";J
19 PRINT "▮▮▮▮COLORE CARATTERI: ";I
21 PRINT "▮▮▮▮▯▯▯▯PREMI UN TASTO PER CONTINUARE"
23 GETKEY A$
25 NEXT I
27 NEXT J
29 NEXT K
```

Il programma ES10 invece richiede i tre numeri colore da utilizzare per il bordo, lo sfondo e i caratteri e mostra la combinazione ottenuta. Per cambiare il colore si deve premere un tasto anche se non compare il messaggio che lo richiede. Variabili utilizzate:
- A$, lettura carattere;
- B, colore bordo;
- S, colore sfondo;
- C, colore carattere.

```
1 REM ES10
3 PRINT "▯▯▯▯▮▮▮PROVA COLORI IN MODO TESTO▯▯▯"
5 INPUT "▮▮COLORE BORDO: ";B
7 INPUT "▮▮COLORE SFONDO: ";S
9 INPUT "▮▮COLORE CARATTERI: ";C
11 COLOR 4,B: REM COLORE BORDO
13 COLOR 0,S: REM COLORE SFONDO
15 COLOR 5,C: REM COLORE CARATTERI
17 PRINT "▯▯▯▯▯▯▮▮▮▮▮PROVA COLORI▯▯▯▯"
19 PRINT "▮▮▮▮▮BORDO: ";B
21 PRINT "▮▮▮▮▮SFONDO: ";S
23 PRINT "▮▮▮▮▮CARATTERI: ";C
25 GETKEY A$
27 GOTO 3
```

Abbiamo preparato il programma ES11 per mostrare qualche esempio dell'istruzione PRINT USING. Il programma contiene alcune costanti numeriche e stringa che vengono visualizzate utilizzando diverse maschere di formato. Le variabili X, Y, Z sono per i dati numerici e le variabili S1$, S2$, S3$ per le stringhe. Si noti che in ogni istruzione di stampa dopo la parola chiave USING puó comparire una sola maschera di formato, che viene utilizzata per tutti i dati della lista di stampa.

```
1 REM ES11
3 DATA 4567,-987,+456,78.956,34567.9,-234
5 DATA -890.76,45678.34
7 S1$="DA CENTRARE"
9 S2$="TROPPO LUNGA"
11 S3$="A DESTRA"
13 PRINT "PROVE DI FORMATI"
15 READ X,Y,Z
17 PRINT USING "##########";X,Y,Z
19 READ X,Y
21 PRINT USING "#######.###↑↑↑↑   ";X,Y
23 READ X,Y,Z
25 PRINT USING "#$##########.###";X
27 PRINT USING "#$##########.###-";Y
29 PRINT USING "$###########.###-";Y
31 PRINT USING "#$##########.###+";Z
33 PRINT USING "**=##################*";S1$
35 PRINT USING "#####";S2$
37 PRINT USING "*########>**";S3$
```

I risultati del programma sono i seguenti:

```
PROVE DI FORMATI
        4567        -987        456
7895600.000E-05     3456790.000E-02
         -$234.000
          $890.760-
$           890.760-
       $45678.340+
*      DA CENTRARE     *
TROPP
*    A DESTRA*
```

Ricordiamo che includendo nelle stringhe di formato caratteri diversi da quelli che hanno particolari significati, questi vengono stampati dove sono; noi abbiamo utilizzato 3 spazi alla fine della stringa di formato per i numeri in formato esponenziale. Si noti il diverso effetto del '$' a seconda che sia o meno preceduto dal carattere '#'. Per comprendere i risultati è opportuno esaminarli uno per uno confrontandoli con le maschere di stampa presenti nelle relative istruzioni.

Il programma ES12 mostra come si puó usare il video come file di input e di output. Alla linea 3 il video viene aperto come file con numero logico 2 e numero di periferica 3. Alle linee 7, 9 e 11, dopo essersi posizionati all'inizio del video e averlo

pulito, vengono stampati: sulla prima riga le 26 lettere dell'alfabeto, sulla seconda un numero e sulla terza un altro numero. A questo punto viene riportato il cursore in posizione HOME con l'istruzione PRINT#2 della linea 17 per poter leggere dal video, e vengono lette con 26 GET#2 le lettere dell'alfabeto, riunendole nella stringa A$. Poi alla linea 23 viene portato il cursore sulla seconda linea del video e vengono lette una per una le cifre e raccolte nella stringa N$, esclusa la prima che sappiamo essere uno spazio, fino a quando si incontra uno spazio (CHR$(32)). Analogo lavoro viene svolto per la terza linea del video raccogliendo i caratteri nella stringa M$.
Alla linea 39 il cursore viene portato sulla tredicesima linea del video e vengono stampati: la prima stringa A$, il valore della seconda con la funzione VAL(N$) e il valore della terza con la funzione VAL(M$).
Il video risulta comunque un file un po' anomalo, infatti ci si posiziona dove si vuole utilizzando il cursore.
Il programma utilizza le seguenti variabili:
- A$, N$, M$, stringhe dei dati;
- B$, N1$, M1$, un carattere;
- K, controllo cicli.

```
1 REM ES12
3 OPEN 2,3: REM APRE IL VIDEO COME FILE
5 REM SCRIVE SULLE PRIME TRE RIGHE DEL VIDEO
7 PRINT#2,"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
9 PRINT#2,0123456789
11 PRINT#2,1234567890
13 REM LEGGE DAL VIDEO LE TRE PRIME RIGHE
15 A$=""
17 PRINT#2,"";:REM A PRIMA RIGA VIDEO
19 FOR K=1 TO 26:GET#2,B$
21 A$=A$+B$:NEXT K
23 PRINT#2,"";:REM A SECONDA RIGA VIDEO
25 N$="":GET#2,N1$
27 GET#2,N1$:IF N1$=CHR$(32) THEN 31
29 N$=N$+N1$:GOTO 27
31 PRINT#2,"";:REM A TERZA RIGA VIDEO
33 M$="":GET#2,M1$
35 GET#2,M1$:IF M1$=CHR$(32) THEN 39
37 M$=M$+M1$:GOTO 35
39 PRINT#2,"" A$
41 PRINT#2,VAL(N$)
43 PRINT#2,VAL(M$)
45 CLOSE 2:STOP
```

2.2 ISTRUZIONI CONDIZIONALI E CICLICHE

L'istruzione condizionale IF risulta piuttosto potente, infatti essa puó essere scritta utilizzando diversi formati. In essi è sempre presente la parte 'IF cond THEN', con il significato che l'espressione condizionale *cond* viene analizzata e se essa risulta *vera* vengono eseguite le istruzioni dopo il THEN, mentre se essa risulta *falsa* vengono eseguite le istruzioni dopo la parola chiave ELSE, se presente, o l'istruzione successiva.
In molte implementazioni del BASIC si ha la limitazione che le istruzioni che seguono rispettivamente le parole chiave THEN e ELSE, se più di una devono essere separate dai ':', ma stare in una sola linea di programma, e la ELSE, anche se preceduta dai ':' deve stare sulla stessa linea. Nel BASIC 7.0 l'introduzione delle parole chiave BEGIN...BEND, che possono comparire dopo THEN e dopo ELSE, consente di utilizzare più linee di programma, ma la parola chiave BEND deve stare sulla stessa linea di ELSE, se presente.
La veridicità di una condizione corrisponde numericamente ad un valore diverso da 0, mentre la sua falsità corrisponde a un valore 0. Segue l'elenco dei diversi formati dell'istruzione IF.

IF cond THEN istruzioni-1

dove le istruzioni dopo THEN possono essere più di una separandole con ':', ma devono stare tutte sulla stessa linea di programma; la fine della linea chiude le istruzioni da eseguire se *cond* risulta *vera*. Se *cond* risulta *falsa* il programma prosegue dalla linea seguente.

IF cond THEN istruzioni-1:ELSE istruzioni-2

dove vale quanto detto prima per *cond vera*, mentre per *falsa* vengono eseguite le istruzioni dopo ELSE. Anche in questo caso deve stare tutto in una linea di programma.

IF cond THEN BEGIN: istruzioni-1: BEND:ELSE BEGIN: istruzioni-2: BEND

dove le due parole chiave BEGIN...BEND delimitano le sequenze di istruzioni da eseguire nei due casi, l'istruzione puó stare su più linee con la limitazione che BEND deve trovarsi sulla stessa linea di ELSE, se presente. Se BEND è sulla stessa linea di 'istruzioni-1' essa deve essere preceduta dai ':'. Come si puó notare dal modo di scriverle, BEGIN e BEND vengono trattate come due istruzioni senza parametri aggiuntivi.

Il programma ES13 mostra il comportamento dell'istruzione IF. Esso chiede in input due numeri M ed N, li confronta e scrive se sono uguali o quale è il maggiore. Si noti che la condizione analizzata risulta vera se N>M, e, in caso questo non sia vero, nella parte ELSE dell'istruzione viene eseguita una seconda istruzione IF senza la parte ELSE per distinguere il caso di uguaglianza.

```
1 REM ES13
3 INPUT N,M
5 IF N>M THEN BEGIN
7 PRINT "N MAGGIORE M":PRINT"N=";N,"M=";M
```

```
9 PRINT "HO ESAURITO LA PARTE PER VERA"
11 BEND:ELSE BEGIN
13 IF M=N THEN PRINT "N UGUALE M":GOTO 17
15 PRINT "N MINORE M"
17 PRINT "N=";N,"M=";M
19 PRINT "HO ESAURITO LA PARTE PER FALSA":BEND
21 PRINT "FINITO"
```

L'istruzione FOR...TO...STEP/NEXT consente di eseguire cicli controllati in modo enumerativo; si deve tener presente che il ciclo viene percorso almeno una volta, infatti il controllo viene eseguito dopo le istruzioni del ciclo. E' possibile eseguire cicli nidificati, cioè uno interno all'altro, ma essi non possono essere intrecciati, cioè devono essere completamente uno interno all'altro e deve terminare prima il più interno, quello aperto per ultimo.

Abbiamo preparato il programma ES14 per esemplificare i cicli FOR nidificati; viene creata la matrice tridimensionale di numeri, A(5,3,4), estraendo dei numeri pseudo-casuali con la funzione RND(0), tali numeri vengono trasformati con un semplice calcolo per farli rientrare nell'intervallo 1/999. Poi la matrice viene visualizzata affiancando i 4 piani, che dipendono dal terzo indice, riga per riga; i punti esclamativi separano i piani. Se si vogliono vedere bene le righe si deve selezionare il video a 80 colonne.

```
1 REM ES14
3 REM STAMPA MATRICE TRIDIMENSIONALE
5 REM DI NUMERI PSEUDO CASUALI
7 REM COMPRESI TRA 1 E 999
9 DIM A(5,3,4)
11 REM CREAZIONE MATRICE
13 FOR J=1 TO 4
15 FOR L=1 TO 3
17 FOR K=1 TO 5
19 X=RND(0):X=INT(X*999+1)
21 A(K,L,J)=X
23 NEXT K,L,J
25 REM STAMPA MATRICE
27 FOR K=1 TO 5
29 FOR J=1 TO 4
31 FOR L=1 TO 3
33 PRINT USING "####";A(K,L,J);
35 NEXT L:PRINT "!";
37 NEXT J:PRINT
39 NEXT K
```

Per provare i cicli DO...LOOP, abbiamo preparato il programma ES15, nel quale vengono chiesti tre numeri e letti nelle variabili X, Y e Z, poi i numeri vengono confrontati e, a seconda dei risultati dei confronti, sono eseguiti cicli controllati in modo diverso mediante le clausole UNTIL e WHILE.

Provando il programma con numeri diversi si puó verificare quali sono le differenze di comportamento e cosa cambia analizzando la condizione prima o dopo LOOP. Inoltre il programma utilizza l'istruzione IF con BEGIN...BEND.

```
1 REM ES15
3 REM PROVA CICLI DO..LOOOP
5 INPUT "SCRIVI TRE NUMERI ";X,Y,Z
6 OPEN 4,4:CMD 4
7 PRINT "X=";X,"Y=";Y,"Z=";Z:PRINT
9 IF X<Z THEN BEGIN:PRINT "X<Z"
11 DO:PRINT X;:X=X+1: LOOP UNTIL X>=Z
13 PRINT:BEND:ELSE BEGIN
15 PRINT "X>=Z"
17 DO UNTIL Z>X:PRINT Z;:Z=Z+1:LOOP:BEND
19 PRINT:PRINT
21 PRINT "X=";X,"Y=";Y,"Z=";Z:PRINT
23 IF X<Y THEN BEGIN:PRINT "X<Y"
25 DO:PRINT X;:X=X+1: LOOP WHILE X<=Y
27 PRINT:BEND:ELSE BEGIN
29 PRINT "X>=Y"
31 PRINT:DO WHILE Y<=X:PRINT Y;:Y=Y+1:LOOP:BEND
33 PRINT
35 PRINT "FINITO"
40 PRINT#4:CLOSE4
```

Per uscire dai cicli al verificarsi di una particolare condizione puó essere usata l'istruzione EXIT.

2.3 SOTTOPROGRAMMI E FUNZIONI

I sottoprogrammi in BASIC fanno parte del programma principale, iniziano a una linea scelta dall'utente e possono cominciare con qualunque istruzione anche se, per rendere più leggibile il programma, si raccomanda che questa sia un'istruzione REM che segnali il punto di entrata. Un sottoprogramma termina con l'istruzione RETURN.
Prima di descrivere cosa fa RETURN dobbiamo chiarire cosa succede quando viene richiamato un sottoprogramma. L'istruzione:

GOSUB num-linea

fa proseguire il programma da *num-linea*, ma prima memorizza nell'area stack del BASIC le informazioni necessarie ad individuare l'istruzione successiva alla GO-SUB (che puó essere la prima di una nuova linea o trovarsi insieme ad altre istruzioni nella linea corrente).
L'area stack del BASIC è una zona di memoria gestita con la tecnica della pila mediante un puntatore all'ultimo elemento introdotto, che sarà il primo ad essere prelevato.
Quando al termine di un sottoprogramma viene incontrato il RETURN, il sistema

va a prelevare dall'area stack l'indirizzo di ritorno e prosegue dall'istruzione successiva al GOSUB.
Se durante l'esecuzione del programma viene incontrato un RETURN, che non è stato preceduto da un GOSUB, il sistema segnala errore. Ogni sottoprogramma deve essere abbandonato con un RETURN, altrimenti si rischia di superare la capacità dell'area stack, che viene riempita e mai svuotata. Per una spiegazione più dettagliata del concetto si veda l'area stack del processore nel capitolo 4.

Dal momento che i sottoprogrammi BASIC fanno parte del programma è importante che prima dell'inizio della zona a loro dedicata compaia un'istruzione che blocca l'esecuzione del programma, come STOP, GOTO o END . Una buona tecnica è quella di raggruppare tutti i sottoprogrammi o all'inizio o alla fine del programma; se si vogliono raggruppare all'inizio la prima istruzione del programma sarà un GOTO che manda a dopo il gruppo dei sottoprogrammi.

Non si presentano problemi riguardo alle variabili; i sottoprogrammi usano le variabili chiamandole con il loro nome e queste sono in comune con il programma principale.

I sottoprogrammi in linguaggio macchina sono esterni al programma BASIC, cioè sono situati in zone prefissate di memoria e il programma BASIC comunica con essi in diversi modi. Per questo argomento rimandiamo al paragrafo 4.1; inoltre nel paragrafo 2.4, abbiamo riportato degli esempi di programmi Basic che utilizzano sottoprogrammi in linguaggio macchina e mostriamo come si effettua la comunicazione dei dati.

Le funzioni possono essere di due tipi:
– definite dal sistema, e quindi esterne al programma, chiamate funzioni intrinseche;
– definite all'inizio del programma, o comunque prima di utilizzarle.
Caratteristica delle funzioni è di restituire un valore o di produrre un azione, quindi non si usano mai da sole, ma all'interno di un'istruzione. Quando una funzione viene richiamata si pongono tra parentesi uno o più parametri, che rappresentano i dati sui quali la funzione lavora per produrre il risultato.
Quando vengono definite le funzioni utente devono essere elencati i parametri formali che vengono utilizzati nella definizione, chiamati parametri *dummy* (*falsi*), in quanto ad essi verranno sostituiti quelli veri al momento del richiamo.

Abbiamo preparato il programma ES16 per mostrare l'utilizzo di sottoprogrammi, che abbiamo raggruppato in fondo al programma, e di funzioni sia intrinseche che definite dal programmatore.
Il programma prepara un vettore di numeri con la funzione RND, chiedendo all'inizio il seme negativo da cui far partire la sequenza; in tale modo se si risponde con lo stesso numero negativo, si ottiene sempre la stessa sequenza. Il programma chiede il seme negativo e il numero dei numeri da trattare. Abbiamo preparato tre sottoprogrammi di ordinamento, che operano sempre con gli stessi numeri, che vengono conservati in disordine. All'inizio di ogni sottoprogramma, sia di estrazione che di ordinamento, viene rilevato il tempo con la funzione TI e così pure alla fine, in modo da poter calcolare il tempo dell'operazione. Abbiamo definito una funzione che serve per ridurre i numeri al massimo a 4 cifre; essi diventano interi perché vengono assegnati a variabili intere.
Il programma chiede se si desidera solo la stampa dei tempi o anche dei dati. L'output è su stampante.

All'inizio il programma pone a zero la variabile SW, alla fine della prima elaborazione, se SW=0 ne viene ripetuta una seconda, dopo essere passati nel modo FAST e aver posto SW=1; risulta così possibile fare un confronto sui tempi di esecuzione dei diversi sottoprogrammi tra loro e vedere il vantaggio dell'esecuzione in FAST.
Le variabili utilizzate sono:
- S$, stringa di 8 spazi;
- X$, Y$, A$, B$, C$, messaggi;
- R$, risposte;
- N$, preparazione stampa;
- SW, controllo esecuzione;
- S, seme negativo per estrazione numeri;
- N, numero dei numeri;
- N%(N), vettore per i numeri da ordinare;
- P%(N), vettore per i numeri da conservare;
- X,IN, variabili di lavoro;
- K, IR, JR, L, J, controllo cicli;
- A, B, calcolo tempo;
- C%, per scambio dati.

Il metodo di ordinamento 1 è quello chiamato *a bolle*, che consiste nel confrontare i numeri a coppie facendo risalire i più piccoli e mandando in fondo i più grandi. All'inizio di ogni ispezione del vettore viene azzerata la variabile J; essa viene posta al valore dell'indice dell'ultimo dato che ha prodotto uno scambio, infatti da quel punto in poi il vettore risulta in ordine. Quando un'ispezione non ha dato luogo a scambi il vettore è ordinato. Non risulta noto a priori il numero dei cicli effettuati.
Il metodo 2, chiamato *a cicli fissi*, confronta con N-1 cicli, il primo elemento con tutti gli altri, e, alla fine pone nella prima posizione il numero minore. A ogni ciclo viene abbandonata una posizione, partendo dall'indice minore.
Il metodo 3, chiamato *a dimezzamento dell'intervallo* confronta gli elementi a coppie, ma con intervallo di N/2, N/4, fino ad arrivare a distanza 1.

```
1 REM ES16
3 REM ORDINAMENTO CON CALCOLO TEMPI
5 REM GENERAZIONE N NUMERI A CASO
7 REM PRIMO RND ARGOMENTO NEGATIVO A SCELTA
9 S$="        ":X$="ORDINATI CON METODO "
11 Y$="ORDINAMENTO":SW=0
13 DO:INPUT "BASE NEGATIVA RND: ";S:LOOP UNTIL S<0
15 DO:INPUT "QUANTI NUMERI: ";N:LOOP UNTIL N>0
17 OPEN 4,4:CMD 4:PRINT "BASE RND: ";S
19 PRINT "ORDINAMENTO DI ";N;" NUMERI":PRINT
21 PRINT#4:CLOSE 4
23 REM NUMERI MINORI UGUALI A 9999
25 INPUT "STAMPA SOLO TEMPI (S/N): ";R$
27 IF R$<>"S" AND R$<>"N" THEN 25
29 REM DIMENSIONA VETTORI
31 DIM N%(N),P%(N)
33 REM DEFINISCE FUNZIONE PER ESTRARRE NUMERI
35 DEF FNY(X)=X*9999+1
```

```
37 REM A E B PER TEMPO INIZIALE E FINALE
39 A=TI
41 N%(1)=FNY(RND(S)):P%(1)=N%(1)
43 FOR K=2 TO N
45 N%(K)=FNY(RND(1)):P%(K)=N%(K)
47 NEXT K:B=TI
49 A$="ESTRATTI":B$="ESTRAZIONE":GOSUB 79
51 C$=" 1":A$=X$+C$:B$=Y$
53 GOSUB 101:GOSUB 79
55 C$=" 2":A$=X$+C$:GOSUB 157
57 GOSUB 135:GOSUB 79
59 C$=" 3":A$=X$+C$:GOSUB 157
61 GOSUB 117:GOSUB 79
63 IF SW=1 THEN SLOW:GOTO 71
65 FAST: SW=1:REM ESEGUE IN FAST
67 OPEN 4,4:CMD 4: PRINT "ESECUZIONE IN FAST"
69 PRINT#4:CLOSE 4:GOTO 39
71 PRINT "BYTE LIBERI: "
73 PRINT "BANCO 0: ";FRE(0)
75 PRINT "BANCO 1: ";FRE(1)
77 STOP
79 REM SOTTOPROGRAMMA STAMPA RISULTATI
81 OPEN 4,4:CMD 4
83 PRINT "TABELLA NUMERI " A$:PRINT
85 IF R$="S" THEN 97
87 IR=0:FOR JR=1 TO N
89 N$=STR$(N%(JR)):N$=LEFT$(N$+S$,8)
91 PRINT N$;:IR=IR+1
93 IF IR=5 THEN PRINT:IR=0
95 NEXT JR:PRINT
97 PRINT "TEMPO " B$;(B-A)/60;" SEC."
99 PRINT#4:CLOSE 4:RETURN
101 REM SOTTOPROGRAMMA METODO A BOLLE
103 L=N-1:A=TI
105 J=0:FOR K=1 TO L
107 IF N%(K)<=N%(K+1) THEN 111
109 C%=N%(K):N%(K)=N%(K+1):N%(K+1)=C%:J=K
111 NEXT K
113 IF J=0 THEN B=TI:RETURN
115 L=J:GOTO 105
117 REM SOTTOPROGRAMMA METODO DIMEZZAMENTO INTERVALLO
119 IN=INT(N/2+0.5):A=TI
121 J=0:FOR K=1 TO (N-IN)
123 IF N%(K)<=N%(K+IN) THEN 127
125 C%=N%(K):N%(K)=N%(K+IN):N%(K+IN)=C%:J=1
127 NEXT K
129 IF IN=1 AND J=0 THEN B=TI:RETURN
131 IF IN=1 THEN 121
```

```
133 IN=INT(IN/2+0.5):GOTO 121
135 REM SOTTOPROGRAMMA METODO CICLI FISSI
137 K=1:A=TI
139 J=K
141 FOR L=J+1 TO N
143 IF N%(J)<=N%(L) THEN 147
145 J=L
147 NEXT L
149 IF K=J THEN 153
151 C%=N%(J):N%(J)=N%(K):N%(K)=C%
153 K=K+1:IF K=N THEN B=TI:RETURN
155 GOTO 139
157 REM SOTTOPROGRAMMA TRASFERIMENTO VETTORE
159 FOR K=1 TO N:N%(K)=P%(K):NEXT K:RETURN
```

Puó essere interessante eseguire il programma ORDINAMENTO, pubblicato nel libro COMMODORE 64 IL BASIC, con il calcolatore in modo 64. Si vede che i tempi di esecuzione sono gli stessi che con il COMMODORE 64, mentre in modo 128 (con il programma ES16) i tempi nell'esecuzione in SLOW sono circa una volta e mezzo. In modo 128 FAST i tempi sono circa la metà rispetto al modo SLOW. L'aumento dei tempi tra modo 64 e modo 128 SLOW dipende dal tempo necessario per la gestione dei banchi di memoria.

2.4 VIDEO A 80 COLONNE IN MODO TESTO

Il chip 8563 consente di utilizzare il video a 80 colonne; esso dispone di 16K di memoria RAM non presi in considerazione negli schemi del capitolo 3, dove peró abbiamo descritto i 37 registri del chip.

I 16K di memoria del chip 8563, quando il calcolatore funziona in modo testo sono utilizzati nel seguente modo:

– da 0 a 1999 ($000-$07CF): mappa video per i codici dei 2000 caratteri dello schermo (80*25=2000);
– da 2048 a 4047 ($0800-$0FCF): mappa attributi dei 2000 caratteri;
– 8192 a 16383 ($2000-$3FFF): descrizione dei caratteri dei due set occupando per ogni carattere 16 byte, 8 byte per la descrizione per punti, più 8 byte normalmente inutilizzati.
Ogni byte della mappa degli attributi contiene:
– bit 7: 0 per set maiuscolo/grafico, 1 per set minuscolo/maiuscolo;
– bit 6: 0 per RVS OFF, 1 per RVS ON;
– bit 5: 0 per normale, 1 per sottolineatura;
– bit 4: 0 per FLASH OFF, 1 per FLASH ON;
– bit 3-0: codice del colore.

Ai 37 registri del chip 8563 non si puó accedere direttamente, ma bisogna servirsi dei due registri del blocco di I/O, attraverso i quali passa la comunicazione. Tali registri sono quelli di indirizzo 54784 ($D600) e 54785 ($D601); il primo viene utilizzato per contenere il numero del registro (indirizzo del registro da 0 a 36) del

chip 8563 con il quale si vuole comunicare. Dopo aver posto nel registro 54784 il numero, da 0 a 36, del registro del chip, bisogna attendere che i suoi due bit di posizione 6 e 7 vadano a 1; a questo punto si puó memorizzare un dato nel registro 54785 e questa operazione produce il passaggio del dato anche nel registro il cui indirizzo era stato posto in 54784. Analogamente per leggere il contenuto del registro del chip, il cui indirizzo era stato posto in 54784, basta prelevare il contenuto del registro 54785.

Abbiamo preparato una routine in linguaggio macchina, servendoci del MONITOR, e l'abbiamo memorizzata sul disco con il nome RR8563, per poter scrivere in uno dei 36 registri o leggere da esso. La routine ha due punti di entrata, uno per leggere e l'altro per scrivere. Abbiamo inoltre preparato un programma in BASIC che, utilizzando tale routine, stampa il contenuto dei 36 registri, chiede se si vuole modificare un registro e, in caso di risposta affermativa, chiede il dato da scrivere nel registro e ve lo scrive.

Prima di scrivere la routine in linguaggio macchina abbiamo deciso come comunicare con essa. Per leggere utilizziamo la routine tramite la funzione USR, cioè svolgiamo i seguenti passi nel programma BASIC:
– scriviamo nei byte 4633 e 4634 l'indirizzo del punto di entrata della routine in linguaggio macchina, nella forma byte LOW, byte HIGH;
– usiamo come parametro nella chiamata di USR il numero del registro del chip da leggere, tale numero viene posto dal sistema nell'accumulatore FAC1 (utilizzato per l'esecuzione dei calcoli dei numeri in formato esponenziale), per cui la routine in linguaggio macchina dovrà ricavarlo da FAC1;
– USR ci fornisce in uscita il contenuto del registro, che possiamo visualizzare o trasferire in una variabile del programma BASIC.

Per contro la routine in linguaggio macchina deve svolgere i seguenti passi:
– selezionare il banco 15 ponendo 0 in $FF00;
– convertire il contenuto di FAC1 in un indirizzo, cosa ottenibile utilizzando la routine del sistema GETADR (indirizo 44815 ($AF0C)), che pone il risultato nei due byte 22 e 23 ($16 e $17);
– porre tale indirizzo in 54784 ($D600);
– attendere che i bit 6 e 7 diventino 1;
– trasferire il contenuto del registro 54785 ($D601) nel registro Y, porre zero nel registro A (accumulatore) e utilizzare la routine del sistema GIVAYF (indirizzo 44803 ($AF03)) per trasformare il contenuto dei due registri A e Y in un numero floating point in FAC1 (infatti USR prende il suo risultato da FAC1).

Per scrivere utilizziamo la routine RR8563 richiamandola con l'istruzione SYS nel formato con due parametri, i cui valori al momento dell'esecuzione vanno rispettivamente nell'accumulatore A e nel registro X del processore. I due parametri sono il numero del registro del chip 8563 nel quale vogliamo scrivere e il dato da scrivere. La routine RR8563 dovrà in conseguenza:
– selezionare il banco 15 ponendo 0 in $FF00;
– trasferire il contenuto di A nel registro 54784 ($D600);
– attendere che i bit 6 e 7 diventino 1;
– trasferire il contenuto del registro X nel registro 54785 ($D601).

Prima di scrivere la routine RR8563, per la quale ora abbiamo tutti gli elementi, dobbiamo decidere dove metterla; abbiamo scelto il buffer della cassetta che inizia

all'indirizzo 2816 ($0B00). Segue la routine in mini-assembler, con aggiunti a destra i commenti, che peró non carichiamo quando la scriviamo con il comando A del MONITOR.

```
F0B00   LDY   #$00    ;punto d'entrata per scrittura registro
F0B02   STY   $FF00   ;selezione del banco 15
F0B05   STA   $D600   ;A va in registro 54784
F0B08   BIT   $D600   ;controlla bit 6 e 7 a 1
F0B0B   BPL   $0B08   ;attende che sia vero
F0B0D   STX   $D601   ;X va in registro 54785
F0B10   RTS           ;uscita della routine
F0B11   LDA   #$00    ;punto d'entrata per lettura registro
F0B13   STA   $FF00   ;selezione del banco 15
F0B16   JSR   $AF0C   ;conv. FAC1 in indirizzo in $16 e $17
F0B19   LDA   $16     ;indirizzo da $16 in 54784
F0B1B   STA   $D600
F0B1E   BIT   $D600   ;controlla bit 6 e 7 a 1
F0B21   BPL   $0B1E   ;attende che sia vero
F0B23   LDY   $D601   ;contenuto registro va in Y
F0B26   LDA   #$00    ;zero va in A
F0B28   JMP   $AF03   ;conv. A e Y in FAC1 e esce
```

Dopo aver scritto la routine, abbiamo utilizzato il comando del MONITOR:
S "RR8563" 8 F0B00 F0B2B
per memorizzarla in un file binario su disco; la routine termina nel byte di indirizzo F0B2A, ma per memorizzarla in un file binario si deve aumentare tale indirizzo di 1.

Riportiamo ora il programma REG8563 che utilizza tale routine e consente di leggere e scrivere i registri del chip 8563.

```
1 REM REG8563
3 REM CARICA ROUTINE LINGUAGGIO MACCHINA
5 REM NEL BUFFER CASSETTA DA 2816 A 2833
7 BLOAD"RR8563",D0,U8,B15,P2816
9 REM LEGGE CONTENUTO 37 REGISTRI
11 REM SCRIVE IND. PER USR IN 4633 E 4634
13 POKE 4633,17:POKE 4634,11
15 PRINT "CONTENUTO REGISTRI 8563":PRINT
17 FOR K=0 TO 36
19 PRINT "REGISTRO ";K,USR(K)
21 NEXT K:PRINT
23 REM SCRITTURA NEI REGISTRI CHIP 8563
25 PRINT "VUOI SCRIVERE NEI REGISTRI"
27 INPUT "DEL CHIP 8563? (S/N) ";R$
29 IF R$="N" THEN STOP
31 INPUT "SCRIVERE NEL REGISTRO: (0/36) ";N
33 INPUT "IL DATO: ";D
35 REM SCRIVE NEL REGISTRO IL DATO
37 SYS 2816,N,D
39 GOTO 25
```

Il programma alla linea 7 carica in memoria il file binario che contiene la routine in linguaggio macchina, poi inserisce in 4633 e 4634 l'indirizzo di inizio della parte lettura di RR8563, che è 2833 ($0B11, byte LOW $11 e byte HIGH $0B, in decimale 17 e 11 rispettivamente). Alle linee 15/21 con un ciclo legge il contenuto dei 37 registri con USR(K) e lo visualizza. Alla linea 25 chiede se si vuole scrivere in qualche registro e, se la risposta è affermativa, chiede il numero del registro e il dato; alla linea 37 va ad eseguire con SYS 2816, N, D la routine per scrivere (punto di entrata 2816 ($0B00)).

La routine RR8563 ci puó essere utile anche per scrivere o leggere la memoria RAM del chip 8563; infatti ci sono tre registri tra i 37 del chip che ci permettono di comunicare con i 16K di memoria RAM, e i registri del chip si manipolano con la routine RR8563. I registri 18 e 19 del chip possono essere caricati con un indirizzo della memoria RAM, questa volta nel formato byte HIGH in 18 e byte LOW in 19, e ci permettono di comunicare con l'indirizzo della memoria RAM selezionato tramite il registro 31, che puó essere letto e scritto. I registri 18 e 19 sono molto utili perché sono autoincrementanti, cioè dopo l'esecuzione dell'operazione l'indirizzo in essi contenuto si incrementa di 1 in modo automatico; in conseguenza si possono facilmente operare trasferimenti di blocchi di memoria.

Come prima applicazione abbiamo preparato il programma ES17 per andare a scrivere direttamente nella mappa video e nella mappa colore del chip 8563. Desideriamo scrivere nelle prime righe del video i 256 caratteri del primo set, il maiuscolo/grafico, 128 in campo diretto e 128 in campo inverso, utilizzando i Display Code dei caratteri, cioè i codici da 0 a 256. Scriviamo una prima volta i caratteri, senza modificare gli attributi del colore, poi cambiamo il colore dei caratteri scritti andando a scrivere un nuovo codice colore negli attributi. La prima volta, dopo aver pulito lo schermo con il carattere CLEAR/HOME scriviamo i codici dei caratteri partendo dall'indirizzo 0 ($0000), a cui corrispondono i byte degli attributi che iniziano all'indirizzo 2048 ($0800).
Dopo scendiamo alla sesta riga del video, cioè utilizziamo l'indirizzo 400 ($190) della mappa video, andiamo a porre a 1 il bit 7 nelle corrispondenti posizioni degli attributi a partire dall'indirizzo 2448 ($0990), cosa che fa passare al secondo set di caratteri, quello minuscolo/maiuscolo, e scriviamo di nuovo i codici da 0 a 255. Questa volta sul video compaiono i caratteri del secondo set in campo diretto e inverso.
A questo punto facciamo scendere il cursore verso il basso, passiamo al set maiuscolo/grafico con PRINT CHR$(142) e visualizziamo una frase con PRINT, poi passiamo al set minuscolo/maiuscolo con PRINT CHR$(14) e visualizziamo una seconda frase. Il programma è terminato e sul video sono presenti caratteri dei due set, comportamento diverso da quello del chip 8564. Si noti inoltre che il programma ES17 viene eseguito in modo FAST, ma il quadro video non scompare.
La routine del sistema di indirizzo 52750 ricarica i caratteri dei due set standard nella zona di memoria del chip ad essi dedicata.

```
1 REM ES17
3 FAST:PRINT"J";:REM PULISCE IL VIDEO
5 PRINT CHR$(142):REM SET MAIUSCOLO/GRAFICO
7 REM SCRIVE CON POKE NELLA ZONA VIDEO
9 REM UTILIZZANDO ROUTINE RR8563
11 BLOAD "RR8563",D0,U8,B15,P2816
13 REM PONE NEI REGISTRI 18 E 19 IL PRIMO
```

```
15 REM INDIRIZZO DELLA MAPPA VIDEO $0000
17 SYS 2816,18,0: REM ZERO NEL BYTE ALTO
19 SYS 2816,19,0:REM ZERO NEL BYTE BASSO
21 REM SCRIVE I CARATTERI CON DISPLAY CODE
23 FOR K=0TO255:REM DA 0 A 255
25 SYS 2816,31,K:REM SCRIVE IL DATO
27 NEXT K
29 REM MODIFICA IL COLORE DEI CARATTERI
31 REM GLI ATTRIBUTI INIZIANO IN $0800
33 SYS 2816,18,8:REM 8 NEL BYTE ALTO
35 SYS 2816,19,0:REM ZERO NEL BYTE BASSO
37 FOR K=0 TO 255
39 SYS 2816,31,2
41 NEXT K
43 SYS 2816,18,9:REM INDIRIZZO $0990
45 SYS 2816,19,144
47 FOR K=0 TO 255
49 SYS 2816,31,130:REM PONE A 1 IL BIT 7
51 NEXT K:REM NELLA MAPPA COLORE
53 SYS 2816,18,1:REM INDIRIZZO $0190
55 SYS 2816,19,144
57 FOR K=0 TO 255
59 SYS 2816,31,K
61 NEXT K
63 REM PASSA AL SET MAIUSCOLO/GRAFICO
65 PRINT"[illegible]"
67 PRINT CHR$(142)
69 PRINT "SONO PASSATO IN MAIUSCOLO GRAFICO"
71 REM PASSA AL SET MINUSCOLO/MAIUSCOLO
73 PRINT CHR$(14)
75 PRINT "♥ONO PASSATO IN MINUSCOLO/MAIUSCOLO"
77 BANK 15:SYS 52750,0:REM RIPRISTINA SITUAZIONE
79 PRINT CHR$(142):SLOW:STOP
```

La descrizione dei caratteri si trova dall'indirizzo 8192 ($2000) della memoria del chip 8563 in poi. Il primo carattere descritto è '@', che corrisponde al Display Code 0. Abbiamo preparato un programma che va a sostituire alla descrizione per punti di '@' da descrizione di un 'omino' e poi visualizza tre omini per riga per tre volte.
Anche questa volta utilizziamo la routine RR8563 per comunicare con la memoria RAM del chip 8563.
Disegnamo l'omino utilizzando degli asterischi, e lo codifichiamo:

```
* *** *      128+32+16+8+2=186
** * **      128+64+16+4+2=214
 *****       64+32+16+8+4=124
  ***        32+16+8=56
  ***        32+16+8=56
 ** **       64+32+8+4=108
**   **      128+64+4+2=198
```

questi numeri li carichiamo nel programma con una linea DATA.

Il programma ES18 è abbastanza semplice:
- carica la routine RR8563;
- carica nei registri 18 e 19 l'indirizzo iniziale del carattere '@', che è $2000;
- con un ciclo percorso 8 volte chiama la routine RR8563 con l'istruzione SYS passando come parametri l'indirizzo 31 e il codice da scrivere nella memoria dei caratteri;
- stampa su tre righe tre caratteri '@', ottenendo invece la visualizzazione di tre 'omini';
- quando si preme un tasto viene rispristinata la situazione iniziale.

```
1 REM ES18
3 REM RIDEFINISCE IL CARATTERE @
5 REM COME UN OMINO
7 DATA 186,214,124,56,56,108,198,0
9 REM CARICA ROUTINE RR8563
11 BLOAD"RR8563",D0,U8,B15,P2816
13 REM CARICA NEI REGISTRI 18 E 19 IL NUMERO $2000
15 REM INDIRIZZO INIZIALE DESCRIZIONE CARATTERI
17 SYS 2816,18,32:SYS 2816,19,0
19 REM SOSTITUISCE L'OMINO AL CARATTERE @
21 FOR K=1 TO 8:READ X
23 SYS 2816,31,X
25 NEXT K
27 REM STAMPA 3 OMINI PER OGNI RIGA
28 REM UTILIZZA SET MAIUSCOLO/GRAFICO
29 PRINT CHR$(142):PRINT "@@@@@@@@@@@@"
30 GETKEY A$:REM ATTENDE TASTO
31 BANK 15:SYS 52750,0:REM RIPRISTINA SITUAZIONE
33 STOP
```

2.5 VIDEO A 40 COLONNE IN MODO TESTO

Quando si lavora in questo modo il sistema preleva la descrizione dei caratteri dalla ROM visibile nel banco 14 da 53248 a 57343 ($D000-$DFFF); in questa zona sono memorizzate le descrizioni dei 128 caratteri del set maiuscolo/grafico in campo diretto (1024 byte) e in campo inverso (1024 byte) e dei 128 caratteri del set minuscolo/maiuscolo in campo diretto e inverso (2048 byte). Alla descrizione dei caratteri il sistema accede in modo trasparente per l'utente quando vengono eseguite le istruzioni che visualizzano caratteri sul video o il sistema emette messaggi. Il chip VIC 8564 provvede alla visualizzazione dei caratteri; esso ricava alcune informazioni dai seguenti byte:

byte 217 ($00D9):
- 0 indica che è attiva la ROM dei caratteri,
- 4 indica che è attivo il blocco di I/O;

byte 2604 ($0A2C):

– bit 7-4: numero che moltiplicato per 1024 dà l'indirizzo di inizio della mappa video,
– bit 3-0: numero che moltiplicato per 1024 dà l'indirizzo di inizio della descrizione dei caratteri in un blocco di 16K, partendo dall'indirizzo zero.

Forzando il byte 217 a contenere 4 il sistema prende la descrizione dei caratteri dalla memoria RAM all'indirizzo indicato dai bit 3-0 del byte 2604.

Abbiamo preparato il programma ES19 per produrre i seguenti effetti:
– spostare l'inizio del programma BASIC a 16384, attivando un modo grafico;
– copiare, lavorando in modo FAST, la descrizione dei caratteri nella memoria RAM a partire dall'indirizzo 8192 nella zona della grafica, che non utilizziamo;
– tornare in modo SLOW e modo testo senza azzerare la pagina grafica;
– definire una finestra video con WINDOW e pulirla;
– modificare i byte 2604 e 217 come descritto sopra per prelevare i caratteri dalla memoria RAM all'indirizzo 8192;
– scrivere la frase PROVA CARATTERI;
– creare un ciclo di attesa che consenta di esaminare bene il video;
– azzerare il quinto byte della descrizione dei caratteri del primo gruppo, in modo che si veda la scritta rovinata;
– stampare di nuovo la frase, che appare già rovinata;
– attendere la pressione di un tasto;
– ripristinare la situazione dei caratteri prelevati dalla memoria ROM.

```
1 REM ES19
3 REM RISERVA LA ZONA DEL VIDEO GRAFICO
5 GRAPHIC 2,1
7 FAST:REM PASSA IN MODO FAST
9 REM BANCO LOGICO 14 DA 53248 A 57343
11 REM VA IN BANCO LOGICO 0 DA 8192 A 12287
13 FOR K=53248 TO 57343
15 BANK 14:Y=PEEK(K)
17 BANK 0:POKE K-45056,Y:NEXT K
19 BANK 15:REM RIPRISTINA DEFAULT
21 SLOW:GRAPHIC 0:REM TORNA IN MODO SLOWW E TESTO
23 WINDOW 7,10,22,15,1:REM CREA FINESTRA
25 REM MA NON ABOLISCE LA ZONA DEL VIDEO
27 REM GRAFICO DOVE SI TROVANO I CARATTERI
29 REM PONE 8 AL POSTO DI 4 IN 2604
31 REM 8*1024=8192 INIZIO DESCRIZIONE CARATTERI
33 POKE 2604, PEEK(2604) AND 240 OR 8
35 REM PONENDO 4 IN 217 AVVISA CHE NON ACCEDE
37 REM ALLA ROM DEI CARATTERI
39 POKE 217,4:PRINT "PROVA CARATTERI"
41 FOR K=1 TO 2000: NEXT K:REM RITARDO
43 REM SFASCIA I CARATTERI AZZERANDO IL
45 REM QUINTO BYTE DEL PRIMO GRUPPO
47 FOR K=8192+4 TO 8192+1028 STEP 8
49 POKE K,0:NEXT K
51 PRINT "PROVA CARATTERI"
```

```
53 GETKEY A$:REM ATTENDE UN TASTO PER PROSEGUIRE
55 POKE 217,0:REM RIPRISTINA ROM CARATTERI
57 POKE 2604, PEEK(2604) AND 240 OR 4
59 STOP
```

Provando ES19 si vede che i caratteri provengono dalla memoria RAM, infatti essi vengono modificati; inoltre quando si preme un tasto immediatamente viene ripristinata la situazione normale e le scritte non appaiono più rovinate. Se alla fine del programma si esegue LIST si vede scorrere il programma nella finestrella creata; per eliminare la finestra basta premere RUN/STOP-RESTORE.

Il programma ES19 puó essere modificato sostituendo le linee 1 e da 43 a 49 con le linee che seguono:

```
1 REM ES19BIS
42 REM SCAMBIA CODICI LETTERE MAIUSCOLE
43 REM CON LETTERE MINUSCOLE
44 FOR K=8192 TO 8192+27*8
45 X=PEEK(K):POKE K,PEEK(K+2048)
47 POKE (K+2048),X
49 NEXT K
```

ottenendo il programma ES19BIS, il quale, dopo aver scritto la frase PROVA CARATTERI, scambia nella memoria RAM la descrizione delle lettere maiuscole con quella delle minuscole. Mentre viene fatto questo lavoro si vede la frase che si modifica; poi la seconda frase viene scritta in minuscolo. Premendo un tasto tutto torna maiuscolo.

2.6 GRAFICA CON VIDEO A 40 COLONNE

La mappa colori per il video in modo testo si trova nel banco 15 da 55296 a 56319 ($D800-$DBFF), i primi 1000 nibble contengono il codice del colore del corrispondente carattere. I *numeri colore* utilizzati con l'istruzione COLOR vanno da 1 a 16, ma essi vengono registrati come codici del colore diminuendoli di 1, quindi da 0 a 15. Per cambiare il colore nella mappa colori si deve usare il numero di sorgente 5, che corrisponde all'area dei caratteri.
In modo grafico invece, la mappa dei colori si trova da 7168 a 8191 e i primi 1000 byte contengono due codici di colore rispettivamente nei bit 7-4 e 3-0. Quando si scelgono i modi grafici 1 e 2, che corrispondono alla matrice di punti standard con schermo intero o diviso, i bit 7-4 contengono dopo un RESET il colore della sorgente 1 (primo piano) e i bit 3-0 il colore della sorgente 0 (sfondo). Nei modi grafici 3 e 4, che corrispondono alla matrice di punti multicolore con schermo intero o diviso, i bit 7-4 contengono dopo un RESET il colore della sorgente 1 (primo piano) e i bit 3-0 il colore della sorgente 2 (multicolor 1). La funzione RCLR(K) consente di rilevare il colore della sorgente K.
Nel modo a matrice di punti all'interno di una zona di 8x8 punti del video, corrispondente alla posizione di un carattere in modo testo, si possono usare solo 2 colori diversi, mentre nel modo multicolore se ne possono usare 3 (il colore dello sfondo è comune).

Dal punto di vista grafico il video è un rettangolo largo 320 punti e alto 200 punti. Le coordinate di riferimento X e Y (X orizzontale e Y verticale) sono 0,0 nell'angolo in alto a sinistra e 319,199 nell'angolo in basso a destra; si veda la figura 2.2.

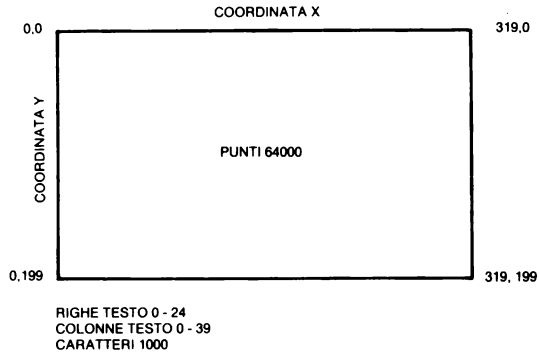


Fig. 2.2 Coordinate grafiche per il video a 40 colonne

La corrispondenza tra la mappa per punti ad alta risoluzione, situata da 8192 a 16383 ($2000-$3FFF), che occupa i primi 8000 byte del blocco, e la visualizzazione è la seguente:
– i byte da 8192 a 8199, cioè i primi 8, forniscono i 64 punti che disegnano la zona in alto a sinistra del primo carattere, e così via, come schematizzato nella figura 2.3.

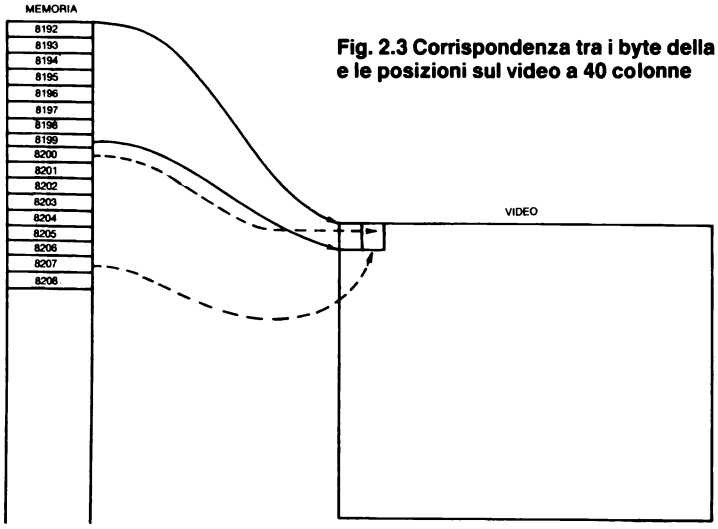


Fig. 2.3 Corrispondenza tra i byte della mappa grafica e le posizioni sul video a 40 colonne

Per questa ragione per disegnare grafici per punti in linguaggio macchina si devono calcolare le coordinate dei punti da disegnare tenendo conto del modo nel quale è utilizzata la memoria della pagina grafica.
Questa preoccupazione non sussiste se si utilizzano le istruzioni grafiche, per le quali si fa riferimento alle coordinate della figura 2.2. Inoltre è disponibile l'istruzione SCALE che consente di fissare delle coordinate ideali, che vengono gestite dal sistema. Se nell'istruzione SCALE si omettono i parametri XMAX e YMAX i valori di default sono 1023 e 1023. Naturalmente si devono fare alcuni calcoli per le coordinate se si è usata l'istruzione SCALE, infatti il sistema deve ridurle sempre al numero di punti disponibili, che sono quelli indicati nella figura 2.2, e considera relativamente al fattore di scala scelto quelle fornite nelle istruzioni grafiche.

Come esempio di utilizzo delle istruzioni grafiche abbiamo preparato il programma ES20, nel quale disegnamo un cerchio e un rettangolo senza aver usato l'istruzione SCALE, poi la utilizziamo e disegnamo ancora un cerchio e un rettangolo più grandi dei precedenti, poi disattiviamo SCALE e circondiamo lo schermo con una cornice; infine coloriamo alcune zone del video e scriviamo un messaggio con l'istruzione CHAR.
Le uniche variabili usate nel programma sono A$ per l'istruzione GETKEY e le due variabili riservate ER e EL, che contengono il numero dell'errore e il numero della linea dove si è verificato l'errore. Abbiamo iniziato il programma con TRAP 57, per vedere, se capita un errore, quale è; questa istruzione in caso di errore manda ad eseguire la linea 57 che ripristina il modo SLOW, il modo testo e stampa la situazione di errore, utilizzando la funzione ERR$. Si noti che la routine di errore termina con RESUME 59, per cui il programma si ferma, dato che in 59 incontra l'istruzione STOP. Nel nostro caso SLOW poteva mancare, infatti non si era passati in FAST, ma scritta così la routine va sempre bene.

```
1 REM ES20
3 REM PROVE GRAFICHE
5 TRAP 57:REM SE ERRORE VA A 2000
7 COLOR 5,1:REM CARATTERI NERI
9 COLOR 0,RCLR(4):REM SFONDO COME CORNICE
11 COLOR 1,3:REM PRIMO PIANO ROSSO
13 PRINT"□":REM PULISCE SCHERMO TESTO
15 GRAPHIC 1,1:REM PASSA IN GRAFICA
17 REM DISEGNA CERCHIO E RETTANGOLO
19 CIRCLE 1,79,99,20
21 BOX 1,230,90,250,110
23 REM CAMBIA SCALA
25 SCALE 1,1023,1023
27 CIRCLE 1,253,508,96
29 BOX 1,718,431,814,586
31 REM TOGLIE FATTORE SCALA
33 SCALE 0
35 REM DISEGNA UNA CORNICE
37 DRAW 1,0,0 TO 319,0 TO 319,199 TO 0,199 TO 0,0
39 DRAW 1,8,4 TO 311,4 TO 311,195 TO 8,195 TO 8,4
41 REM COLORA PARTI DI FIGURE
43 PAINT 1,1,1
```

```
45 PAINT 1,100,120
47 PAINT 1,228,87
49 CHAR 1,10,19,"FINITE PROVE GRAFICHE"
51 REM PASSA IN MODO TESTO SE SI PREME UN TASTO
53 GETKEY A$:REM ATTENDE UN TASTO
55 GRAPHIC 0:PRINT"FINITO":END
57 SLOW:GRAPHIC 0:PRINT ERR$(ER);EL:RESUME 59
59 STOP
```

Si noti che per portare sfondo e bordo allo stesso colore si usa il numero colore fornito da RCLR(4). Abbiamo calcolato le coordinate del centro del cerchio e degli estremi della diagonale del rettangolo in modo da centrarli nelle due metà del video. Con l'istruzione SCALE utilizziamo per XMAX e YMAX i due valori di default; scrivendo solo SCALE 1, i risultati non cambiano. Abbiamo calcolato le coordinate da usare con CIRCLE e BOX in modo che i due precedenti disegni risultino interni ai nuovi, utilizzando delle semplici proporzioni. Si noti che per colorare le parti di figura basta prendere un punto interno alla zona da colorare.

Passando al modo grafico multicolore vengono dimezzate le coordinate orizzontali del video, per cui la figura 2.2 va letta con 159 al posto di 319, e i punti sono larghi il doppio. Per mostrare questo comportamento con le istruzioni grafiche, abbiamo preparato il programma ES21, che, dopo esser passato in modo grafico multicolore, predispone i colori delle sorgenti e poi disegna con CIRCLE un cerchio nel centro del video, traccia un disegno con DRAW cambiando sorgente di colore per ogni lineetta e poi scrive un messaggio con CHAR.

```
1 REM ES21
3 REM PROVA GRAFICA MULTICOLORE
5 TRAP 43
7 COLOR 0,2:COLOR 4,2:REM SFONDO E CORNICE
9 COLOR 2,3:COLOR 3,6:REM COLORI MULTICOLORE
11 COLOR 1,1:COLOR 5,1:REM PRIMO PIANO E CARATTERI
13 GRAPHIC 3,1:REM GRAFICA MULTICOLORE
15 REM DISEGNA CERCHIO IN CENTRO
17 CIRCLE 2,79,99,10
19 REM TRACCIA UN DISEGNO CON DRAW
21 DRAW 1,10,10 TO 14,14
23 DRAW 2,14,14 TO 18,18
25 DRAW 3,18,18 TO 22,22
27 DRAW 1,22,22 TO 32,22
29 DRAW 2,32,22 TO 42,32
31 DRAW 3,42,32 TO 48,38
33 DRAW 2,48,38 TO 10,10
35 REM SCRIVE CON CHAR
37 CHAR 3,5,19,"FINITO PROVA"
39 GETKEY A$:GRAPHIC 0:PRINT "FINITO"
41 END
43 SLOW:GRAPHIC 0:PRINT ERR$(ER);EL:RESUME 45
45 STOP
```

Si notino le coordinate utilizzate per il centro del cerchio.

Per utilizzare la pagina grafica per punti in modo multicolore è necessario utilizzare due bit per ogni *punto colore*, due bit consentono di memorizzare un numero da 0 a 3, che è il numero della sorgente del colore da utilizzare per colorare due punti vicini. In conseguenza vengono utilizzati 40 byte della pagina grafica in modo multicolore per riempire di punti, a due a due dello stesso colore, un'intera linea. Abbiamo preparato il programma ES22 per mostrare questo comportamento. Il programma esegue i seguenti passi:
– predispone la routine di errore;
– pone sfondo e bordo in bianco, primo piano e caratteri in nero, colore 1 multicolore rosso e colore 2 multicolore verde;
– pone nei byte della pagina grafica da 8192 a 8504, con intervallo di 8, che corrispondono alla prima linea di 320 punti della prima riga del video in modo testo, il numero 27, che in binario è 00 01 10 11, e significa 'usare per i primi due punti il colore della sorgente 0, per i due seguenti quello della sorgente 1, per i due seguenti quello della sorgente 2 e per gli ultimi due quello della sorgente 3';
– svolge un lavoro analogo per la prima linea della terza riga del video (in coordinate testo), la cui immagine nella pagina grafica va da 8832 a 9144, con intervallo di 8, utilizzando il numero 25, che rappresenta ordinatamente le sorgenti di colore 0, 1, 2, 1;
– svolge un lavoro analogo per la prima linea della terzultima riga del video (in coordinate testo), indirizzando i byte da 15232 a 15544, con intervallo di 8, e utilizzando con il numero 20 le sorgenti di colore 0,1,1,0;
– svolge un lavoro analogo per la prima linea dell'ultima riga del video (in coordinate testo), indirizzando i byte da 15872 a 16184, con intervallo di 8, e utilizzando con il numero 105 le sorgenti di colore 1,2,2,1;
– attende la pressione di un tasto, torna in modo testo e scrive un messaggio.

```
1 REM ES22
3 REM PROVA GRAFICA MULTICOLORE PER PUNTI
5 TRAP 45
7 COLOR 0,2:COLOR 4,2:REM SFONDO E CORNICE
9 COLOR 2,3:COLOR 3,6:REM COLORI MULTICOLORE
11 COLOR 1,1:COLOR 5,1:REM PRIMO PIANO E CARATTERI
13 GRAPHIC 3,1:REM GRAFICA MULTICOLORE
15 REM PREPARA PUNTI NELLA PAGINA GRAFICA
17 REM LINEETTE SU PRIMA RIGA DA 8192
19 FOR K=0 TO 39
21 POKE 8192+K*8,27:NEXT K
23 REM LINEETTE SU TERZA RIGA DA 8832
25 FOR K=0 TO 39
27 POKE 8832+K*8,25:NEXT K
29 REM LINEETTE SU TERZULTIMA RIGA DA 15232
31 FOR K=0 TO 39
33 POKE 15232+K*8,20:NEXT K
35 REM LINEETTE SU ULTIMA RIGA DA 15872
37 FOR K=0 TO 39
39 POKE 15872+K*8,105:NEXT K
41 GETKEY A$:GRAPHIC 0:PRINT "FINITO"
43 END
```

```
45 SLOW:GRAPHIC 0:PRINT ERR$(ER);EL:RESUME 47
47 STOP
```

Se si cambiano i numeri che si scrivono nella pagina grafica, utilizzando sempre lo stesso numero di sorgente, si vedono linee continue dello stesso colore; attenzione peró a non usare la sorgente 0, altrimenti non si vede la linea.

Per tracciare grafici si puó utilizzare l'istruzione DRAW disegnando un punto per volta o una lineetta per volta, oppure utilizzare una routine in linguaggio macchina.

Ci occupiamo ora di grafica per punti non in modo multicolore. Abbiamo preparato una routine in linguaggio macchina, di nome GLM2, che ricevendo le coordinate di un punto lo disegna.

Abbiamo deciso di utilizzare un sistema di coordinate che pone il punto 0,0 nell'angolo in basso a sinistra, invece che in alto a sinistra come fa il sistema, in conseguenza, mentre le coordinate X sono invariate, la Y del sistema si calcola eseguendo: 199 - Ynostra. Nel nostro sistema di coordinate il punto 0,0 è in basso a sinistra, l'asse X è orientata verso destra e l'asse Y dal basso verso l'alto. Dobbiamo trovare la relazione che esiste tra la nostra coppia X,Y e il bit corrispondente nella pagina grafica. Per X e Y valgono le relazioni:

0 <= X <= 319 e 0 <= Y <= 199.

Ogni punto individuato da X e Y appartiene a una riga e a una colonna del video in modo testo, cioè sta dentro la posizione di un carattere. Chiamiamo R, la coordinata di riga, da 0 a 24, e C la coordinata di colonna, da 0 a 39; ricordiamo che R=0 significa linea in alto e C=0 colonna a sinistra. Chiamiamo B il numero del byte della posizione carattere a cui appartiene il punto, B è compreso tra 0 e 7. Vale la seguente relazione:

R = INT((199-Y)/8)
C = INT(X/8)
B = 199-Y-R*8

che verifichiamo per il punto di coordinate X=8 e Y=8. Per tale punto si trova: R=INT((199-8)/8)=23, C=INT(8/8)=1, B=199-823*8=7, infatti il punto si trova nell'angolino in basso a sinistra della seconda posizione carattere della penultima riga di testo.
A questo punto dobbiamo trovare le formule che ci danno la possibilità di inserire un bit a 1 nella giusta posizione del byte corrispondente alla posizione individuata sul video. L'indirizzo del primo byte della pagina grafica è 8192, per cui, tenendo conto di quanto detto, l'indirizzo I del byte che ci interessa è:

I = 8192 + R*320 + C*8 + B

resta da determinare come trovare la posizione del punto nel byte B.
Calcolando E = X-C*8 troviamo un numero compreso tra 0 e 7 che ci fornisce la posizione del bit nel byte al contrario; per ottenere la posizione giusta dobbiamo calcolare E = 7 - E e il numero da usare con l'operatore OR è 2↑(7-(X-C*8)).

In conseguenza l'istruzione BASIC che inserisce il punto è:

POKE I, PEEK(I) OR 2↑(7-(X-C*8))

Per quanto riguarda la posizione della routine in linguaggio macchina, abbiamo deciso di abbassare il TOP del BASIC nel banco 0 all'indirizzo 63232; di posizionare la routine a partire dall'indirizzo 63280 ($F730) e di utilizzare come memorie di lavoro i byte da 63450 a 63457 ($F7DA-$F7E1) e i byte di indirizzo 254 e 255 ($FE e $FF) della pagina zero.
Riportiamo il listato di GLM2 ottenuto dirottando l'uscita del MONITOR alla stampante e utilizzando il comando D 0F730 0F7BE.

```
. 0F730  A9 C7     LDA #$C7
. 0F732  38        SEC
. 0F733  ED DF F7  SBC $F7DF
. 0F736  8D DF F7  STA $F7DF
. 0F739  29 F8     AND #$F8
. 0F73B  8D DE F7  STA $F7DE
. 0F73E  AD E0 F7  LDA $F7E0
. 0F741  29 F8     AND #$F8
. 0F743  8D DD F7  STA $F7DD
. 0F746  AD DF F7  LDA $F7DF
. 0F749  29 07     AND #$07
. 0F74B  85 FE     STA $FE
. 0F74D  AD E0 F7  LDA $F7E0
. 0F750  29 07     AND #$07
. 0F752  8D DC F7  STA $F7DC
. 0F755  A5 FE     LDA $FE
. 0F757  18        CLC
. 0F758  6D DD F7  ADC $F7DD
. 0F75B  85 FE     STA $FE
. 0F75D  A9 20     LDA #$20
. 0F75F  6D E1 F7  ADC $F7E1
. 0F762  85 FF     STA $FF
. 0F764  A9 00     LDA #$00
. 0F766  8D DB F7  STA $F7DB
. 0F769  AD DE F7  LDA $F7DE
. 0F76C  18        CLC
. 0F76D  2A        ROL
. 0F76E  8D DA F7  STA $F7DA
. 0F771  2E DB F7  ROL $F7DB
. 0F774  2E DA F7  ROL $F7DA
. 0F777  2E DB F7  ROL $F7DB
. 0F77A  AD DE F7  LDA $F7DE
. 0F77D  6D DA F7  ADC $F7DA
. 0F780  8D DA F7  STA $F7DA
. 0F783  A9 00     LDA #$00
. 0F785  6D DB F7  ADC $F7DB
. 0F788  8D DB F7  STA $F7DB
. 0F78B  18        CLC
. 0F78C  2E DA F7  ROL $F7DA
. 0F78F  2E DB F7  ROL $F7DB
```

```
. 0F792  2E DA F7 ROL $F7DA
. 0F795  2E DB F7 ROL $F7DB
. 0F798  2E DA F7 ROL $F7DA
. 0F79B  2E DB F7 ROL $F7DB
. 0F79E  18       CLC
. 0F79F  AD DA F7 LDA $F7DA
. 0F7A2  65 FE    ADC $FE
. 0F7A4  85 FE    STA $FE
. 0F7A6  AD DB F7 LDA $F7DB
. 0F7A9  65 FF    ADC $FF
. 0F7AB  85 FF    STA $FF
. 0F7AD  A9 80    LDA #$80
. 0F7AF  AE DC F7 LDX $F7DC
. 0F7B2  F0 04    BEQ $F7B8
. 0F7B4  4A       LSR
. 0F7B5  CA       DEX
. 0F7B6  D0 FC    BNE $F7B4
. 0F7B8  A0 00    LDY #$00
. 0F7BA  11 FE    ORA ($FE),Y
. 0F7BC  91 FE    STA ($FE),Y
. 0F7BE  60       RTS
```

La routine GLM2 applica le formule che abbiamo riportato, partendo dai valori di X e Y che trova in memoria, e precisamente, prima di eseguire SYS 63280 è necessario memorizzare in:
– 63455 ($F7DF) il valore di Y, che è al massimo 199;
– 63456 ($F7E0) il valore della parte meno significativa di X, cioè X AND 255;
– 63457 ($F7E1) il valore della parte più significativa di X, cioè X/256 (ovviamente viene memorizzata solo la parte intera).

La routine memorizza nei due byte della pagina zero di indirizzo 254 e 255 l'indirizzo del byte nel quale va inserito il bit a 1 che disegna il punto. Lasciamo al lettore interessato il commento della routine riga per riga, tenendo anche conto del fatto che per poterla leggere bene bisogna aver prima studiato un po' il linguaggio macchina.
La routine GLM2 è stata memorizzata sul disco come file binario e viene richiamata dai programmi che la utilizzano.

Come semplice esempio di utilizzo della routine GLM2 abbiamo preparato il programma ES23, che chiede le coordinate di un punto e lo disegna. In questo programma appaiono alternativamente la pagina grafica e la pagina testo; per tornare al testo basta premere un tasto. Per uscire dal programma premere RUN/STOP-RESTORE.

```
1 REM ES23
3 REM SCRIVE SU PAGINA GRAFICA
5 REM TRAP 500:REM ROUTINE ERRORE
7 REM ABBASSA IL TOP DEL BASIC A 63232
9 REM CARICA IL CODICE MACCHINA DA 63280 A 63422
```

```
11 REM ESADECIMALE $F730-$F7BE
13 REM ABBASSA IL TOP DEL BASIC 247*256=63232
15 POKE 4626,0:POKE 4627,247
17 GRAPHIC 1,1:REM RISERVA PAGINA GRAFICA
19 REM CARICA CODICE MACCHINA DA 63280 IN BANCO 0
21 BLOAD "GLM2",D0,U8,B0,P63280
23 REM TORNA IN MODO TESTO
25 GRAPHIC 0:PRINT"□"
27 REM CHIEDE LE COORDINATE DI UN PUNTO
29 REM E LO DISEGNA
31 REM COORDINATE DEL PUNTO TRA 0-319 E 0-199
33 INPUT "COORDINATE PUNTO: ";X,Y
35 REM MEMORIZZA COORDINATE NEI BYTE RICHIESTI
37 POKE 63455,Y:REM MEMORIZZA Y
39 POKE 63456,X AND 255:REM MEMORIZZA LOW X
41 POKE 63457,X/256:REM MEMORIZZA HIGH X
43 GRAPHIC 1:BANK 0: SYS 63280:REM DISEGNA PUNTO
45 GETKEY A$:REM ATTENDE UN TASTO
47 GRAPHIC 0:PRINT"■":GOTO33
49 SLOW:GRAPHIC 0:PRINT ERR$(ER),EL:RESUME 51
51 STOP
```

Il programma chiede le coordinate X e Y di un punto, le memorizza nei tre byte richiesti e chiama la routine che disegna il punto.

Come ulteriore esempio di grafica per punti abbiamo preparato il programma ES24, che utilizzando la routine GLM2 traccia il grafico della funzione X*SIN(X) tra -7*3.14 e +7*3.14. Il programma è molto simile al precedente, solo che, invece di chiedere le coordinate di un punto, calcola con un ciclo, per X che varia da 0 a 319, i valori di Y e per disegnare ogni punto chiama la routine GLM2. Non abbiamo utilizzato l'istruzione FAST, perché altrimenti non si vede procedere il disegno, ma esso compare solo alla fine.

```
1 REM ES24
3 TRAP 45:REM ROUTINE ERRORE
5 REM DISEGNA FUNZIONE Y=X*SIN(X)
7 REM ABBASSA IL TOP DEL BASIC A 63232
9 POKE 4626,0:POKE 4627,247
11 REM SISTEMA COLORI
13 COLOR 0,7:COLOR 4,7:COLOR 1,2
15 REM ATTIVA LA PAGINA GRAFICA E LA PULISCE
17 GRAPHIC 1,1
19 REM CARICA ROUTINE CHE TRACCIA I PUNTI
21 BLOAD "GLM2",D0,U8,B0,P63280
23 REM CALCOLO FUNZIONE
25 FOR X=0 TO 319:I=14*π*(X/319)-7*π
27 Y=I*SIN(I):Y=Y*90/(7*π)+100
29 GOSUB 33:NEXT X
```

```
31 GOTO 31:REM ATTESA SENZA MESSAGGIO
33 REM ROUTINE CHE TRACCIA UN PUNTO
35 REM DI COORDINATE X E Y
37 POKE 63455,Y:REM MEMORIZZA Y
39 POKE 63456,X AND 255:REM MEMORIZZA LOW X
41 POKE 63457,X/256:REM MEMORIZZA HIGH X
43 BANK 0:SYS 63280:RETURN
45 SLOW:GRAPHIC 0:PRINT ERR$(ER),EL:RESUME 47
47 STOP
```

Abbiamo poi modificato il programma ES24, ottenendo il programma ES25 che segue, nel quale non utilizziamo la routine GLM2 per tracciare i punti, ma invece l'istruzione DRAW. Abbiamo mantenuto invariate le formule di calcolo, ma, dato che l'istruzione DRAW utilizza l'asse delle Y invertita, la curva appare ribaltata.

```
1 REM ES25
3 TRAP 27:REM ROUTINE ERRORE
5 REM DISEGNA FUNZIONE Y=X*SIN(X)
7 REM UTILIZZANDO ISTRUZIONE DRAW
9 REM SISTEMA COLORI
11 COLOR 0,7:COLOR 4,7:COLOR 1,2
13 REM ATTIVA LA PAGINA GRAFICA E LA PULISCE
15 GRAPHIC 1,1
17 REM CALCOLO FUNZIONE
19 FOR X=0 TO 319:I=14*π*(X/319)-7*π
21 Y=I*SIN(I):Y=Y*90/(7*π)+100
23 DRAW 1,X,Y:NEXT X
25 GOTO 25:REM ATTESA SENZA MESSAGGIO
27 SLOW:GRAPHIC 0:PRINT ERR$(ER),EL:RESUME 29
29 STOP
```

Segue il programma ES26, costruito utilizzando ancora la routine GLM2 per disegnare i punti, ma utilizzando un'interessante tecnica per ricevere il testo della funzione da disegnare. Il programma richiede la funzione come una stringa di dati e lavorando sul video e sul buffer della tastiera la inserisce al posto giusto nel programma, come funzione definita dall'utente.
Il programma per realizzare questo scopo svolge i seguenti passi:
– richiede nella stringa F$ la funzione, che deve essere scritta con le notazioni BASIC;
– pulisce il video e si porta sulla quarta riga, dove scrive:
"37 DEF FNY(X)=" F$ cioè fa apparire la definizione della funzione come linea 37 del programma sul video;
– scrive sulla riga seguente:
"GOTO 37" e manda il cursore nella posizione HOME, senza cancellare il video;
– scrive nel buffer della tastiera due volte il codice ASCII di RETURN (13) e pone il numero 2 nel byte 208, puntatore al buffer della tastiera;
– esegue END e quindi il calcolatore risponde con READY e si porta sulla quarta riga video, dove i due RETURN presenti nel buffer della tastiera, vengono eseguiti; il primo inserisce nel programma la linea 37, il secondo fa eseguire in immediato il GOTO 37 e quindi il programma continua dalla linea 37.

Questo programma puó quindi disegnare il grafico della funzione che gli viene fornita; esso richiede i due estremi dell'intervallo da considerare, se si vuole tracciare anche l'asse delle X e se si vuole cancellare la pagina grafica. Prima di iniziare il disegno vengono calcolati il minimo e il massimo della funzione nell'intervallo e calcolata la posizione dell'asse X. La funzione viene disegnata con un ciclo utilizzando la routine GLM2. Alla fine il programma chiede se si vuole disegnare un altro grafico e permette di farlo senza cancellare il precedente.
Le variabili utilizzate sono:
- F$, stringa che costituisce la funzione;
- X, Y, coordinate funzione;
- ID, FD, estremi dell'intervallo;
- A$, risposte;
- AX, flag per disegnare asse X;
- CL, flag per cancellare vecchia mappa;
- IR, FR, AD, AR, ZR, variabili di lavoro;
- ER e EL, variabili di sistema per routine errore.

```
1 REM ES26
3 TRAP 127:REM ROUTINE ERRORE
5 REM SYNTAX ERROR SE FUNZIONE MAL DEFINITA
7 REM ABBASSA IL TOP DEL BASIC A 63232
9 POKE 4626,0:POKE 4627,247
11 REM SISTEMA I COLORI
13 COLOR 0,7:COLOR 4,7:COLOR 1,2
15 REM ATTIVA LA PAGINA GRAFICA E LA PULISCE
17 GRAPHIC 1,1
19 REM CARICA ROUTINE CHE TRACCIA I PUNTI
21 BLOAD"GLM2",D0,U8,B0,P63280
23 GRAPHIC 0:REM TORNA AL TESTO
25 REM INCORPORA LA FUNZIONE NEL PROGRAMMA
27 INPUT "F(X) ";F$
29 PRINT CHR$(147)CHR$(17)CHR$(17)CHR$(17);
31 PRINT CHR$(144)"37 DEF FNY(X)=" F$
33 PRINT "GOTO 37"CHR$(19)
35 POKE 208,2:POKE 842,13:POKE 843,13:END
37 DEF FNY(X)=4*X↑4-3*X↑3+2*X↑2-18
39 PRINT CHR$(154)CHR$(147)
41 INPUT "DOMINIO DA ";ID
43 INPUT "A ";FD
45 PRINT:PRINT "DISEGNO L'ASSE DELLE X ?"
47 GETKEY A$:IF A$<>"S" AND A$<>"N" THEN 47
49 AX=1
51 IF A$="N" THEN AX=0
53 PRINT:PRINT "CANCELLO LA MAPPA ?"
55 GETKEY A$:IF A$<>"S" AND A$<>"N" THEN 55
57 CL=1
59 IF A$="N" THEN CL=0
61 PRINT
63 PRINT "STO CALCOLANDO MASSIMO E MINIMO"
```

```
65 FAST:IR=0:FR=0:AD=FD-ID
67 FOR X=0 TO 319
69 I=ID+X*AD/320
71 Y=FNY(I)
73 IF IR>Y THEN IR=Y
75 IF FR<Y THEN FR=Y
77 NEXT X
79 AR=FR-IR:SLOW
81 REM ATTIVA LA PAGINA GRAFICA
83 IF CL=1 THEN GRAPHIC 1,1:GOTO 87
85 GRAPHIC 1
87 ZR=ABS(IR/AR*199)
89 FOR X=0 TO 319
91 I=ID+X*AD/320
93 Y=FNY(I)*199/AR+ZR
95 GOSUB 115
97 IF AX=1 THEN Y=ZR:GOSUB 115
99 NEXT X
101 POKE 208,0
103 GETKEY A$:REM ATTENDE UN TASTO
105 GRAPHIC 0
107 PRINT CHR$(147)"VUOI UN'ALTRO GRAFICO ?"
109 GETKEY A$:IF A$<>"S" AND A$<>"N" THEN 109
111 IF A$="N" THEN STOP
113 GOTO 27
115 REM ROUTINE CHE TRACCIA UN PUNTO
117 REM DI COORDINATE X E Y
119 POKE 63455,Y:REM MEMORIZZA Y
121 POKE 63456,X AND 255:REM MEMORIZZA LOW X
123 POKE 63457,X/256:REM MEMORIZZA HIGH X
125 BANK 0: SYS 63280:RETURN
127 SLOW:GRAPHIC 0:PRINT ERR$(ER),EL:RESUME 129
129 STOP
```

Il programma consente di tracciare grafici di funzioni senza asintoti verticali.

Si ricordi che è possibile trasferire parti di video grafico in variabili stringa e viceversa con le istruzioni SSHAPE e GSHAPE, utilizzando le coordinate grafiche della diagonale del rettangolo interessato; solo che le stringhe non possono superare 255 caratteri. Inoltre è possibile salvare parti di video in file binari su disco o cassetta, lavorando sugli indirizzi della pagina grafica e della mappa colore, e poi ricaricarli.

2.7 GRAFICA CON VIDEO A 80 COLONNE

Il video a 80 colonne corrisponde a 640 x 200 punti e consente di fare della buona grafica; per una pagina grafica di queste dimensioni occorrono 128000 punti (640*200=128000) e quindi 16000 byte (128000/8=16000), cioè quasi tutta la me-

moria RAM del chip 8563. In conseguenza, utilizzando tutto il video, possiamo fare solo della grafica monocromatica.
L'indirizzo del primo byte, nell'angolo in alto a sinistra, è 0 ($0000) e quello dell'ultimo byte, angolo in basso a destra, è 15999 ($3E7F).

Per poter avere a disposizione tutta la memoria per la grafica dobbiamo porre 128 nel registro 25 del chip 8563 eliminando la presenza della mappa degli attributi.

Per vedere come sono utilizzati i byte della pagina grafica abbiamo preparato il programma ES27, nel quale, dopo aver azzerato tutta la pagina, poniamo 8 bit a 1 a partire dal byte di indirizzo 0 e ogni 648 byte per 25 volte.
Il programma all'inizio carica la routine RR8563, poi, per azzerare i 16000 byte della pagina grafica usiamo il seguente metodo:
- poniamo l'indirizzo 0 nei due registri 18 e 19;
- poniamo 0 nel registro 31;
- poniamo 0 nel registro 30, che funge da contatore e se a zero vale 256;
- eseguiamo 64 volte il ciclo, nel quale ogni volta vengono azzerati 256 byte.

In tale modo viene completamente azzerata la RAM di 16K.
Poi con un ciclo nel quale facciamo variare gli indirizzi da 0 a 15552 con passo 648 disegnamo le 25 lineette, utilizzando i registri 18 e 19 per l'indirizzo e il registro 31 per il dato.
Il programma termina attendendo la pressione di un tasto; prima di arrivare allo STOP viene chiamata la routine che azzera la memoria, viene chiamata la routine in 52750 ($CE0E), passando il valore 0 per A, che carica la memoria con la descrizione dei caratteri, e viene posto 64 nel registro 25 del chip 8563 per riattivare gli attributi e passare in modo testo.

```
1 REM ES27
3 REM GRAFICA AD ALTA RISOLUZIONE
5 BLOAD"RR8563",D0,U8,B15,P2816
7 FAST
9 REM PASSA IN GRAFICA MONOCROMATICA
11 SYS 2816,25,128
13 GOSUB41:REM AZZERA PAGINE
15 REM DISEGNA IN GRAFICA
17 REM DISEGNA 25 LINEETTE DI 8 PUNTI
19 FOR K=0 TO 15552 STEP 648
21 LO=K AND 255
23 HI=INT(K/256)
25 SYS 2816,18,HI:SYS 2816,19,LO
27 SYS 2816,31,255
29 NEXT K:SLOW
31 GETKEY A$:REM ATTESA TASTO
33 GOSUB 41:REM AZZERA PAGINE
35 BANK 15:SYS 52750,0:REM RIPRISTINA SITUAZIONE
37 SYS 2816,25,64:REM RIPRISTINA TESTO
39 PRINT"□":STOP
41 REM ROUTINE AZZERAMENTO MEMORIA
43 REM CARICA INDIRIZZO ZERO NEI REGISTRI 18 E 19
45 SYS 2816,18,0:SYS 2816,19,0
```

```
47 REM PONE A ZERO TUTTE LE PAGINE GRAFICHE
49 FOR K=0 TO 63
51 SYS 2816,31,0:SYS 2816,30,0
53 NEXT K:RETURN
```

La routine per l'azzeramento della memoria si trova nelle linee 41/53.
L'esecuzione di questo programma ci fa comprendere che i byte da 0 a 15999 corrispondono ordinatamente ai punti della finestra video partendo dall'angolo in alto a sinistra, i primi 80 byte corrispondono alla prima riga, e così via. Possiamo concludere che la gestione della pagina grafica del chip 8563 risulta più semplice di quella del chip 8564.

Nel programma ES27 abbiamo disegnato in grafica 8 punti consecutivi, scrivendo nei byte della pagina grafica in posizioni fisse il numero 255 (8 bit a 1, cioè 8 punti accesi). Per poter tracciare dei grafici dobbiamo trovare le formule che ci permettono di inserire un punto (un bit 1) in qualunque posizione di qualunque byte. Tenendo conto che ogni linea è formata da 640 punti, la coordinata X puó variare da 0 a 639, mentre, essendo 200 le linee del video, la coordinata Y puó variare da 0 a 199. Noi desideriamo utilizzare coordinate abituali, cioè avere l'asse Y dal basso verso l'alto, mentre gli indirizzi della pagina grafica sono crescenti dall'alto verso il basso. Se chiamiamo X1 e Y1 le nostre coordinate, con l'origine nell'angolo in basso a sinistra del video, per adattarle alla situazione della pagina grafica dove le coordinate sono X e Y, basta porre:

X = X1 e Y = Y1 - 199.

Chiamiamo I l'indirizzo del byte dove dobbiamo porre a 1 il bit corrispondente al punto di coordinate X e Y. In ogni linea video ci sono 80 byte e l'indirizzo del primo byte (in alto a sinistra) è zero. Abbiamo:

I = INT(Y) * 80 + INT(X/8)

dobbiamo calcolare il resto di X/8, che chiamiamo DX,

DX = X AND 7

tale resto puó variare da 0 a 7, e il numero da utilizzare per inserire un bit 1, che chiamiamo P è dato da:

P = 2 ↑ (7 - DX).

Il numero P deve essere usato con un'istruzione che in BASIC si dovrebbe scrivere:

POKE I,PEEK(I) OR P

ma che dovremo realizzare utilizzando la routine RR8563 tramite i registri del chip.

Abbiamo preparato il programma ES28 per tracciare il grafico di una funzione. Esso risulta una modifica del programma ES26 utilizzato per la grafica con video a 40 colonne. Non ripetiamo le spiegazioni della parte iniziale che chiede all'utente una funzione e la incorpora nel programma, poi chiede l'intervallo per tracciare il grafico e se si desidera l'asse X. In questo caso alla fine si cancella la pagina

grafica e non si puó sovrapporre un altro grafico.
La parte diversa del programma si ha dalla linea 59 in poi; inoltre all'inizio viene letta la routine RR8563 invece dell'altra.

```
1 REM ES28
3 REM GRAFICA CON VIDEO A 80 COLONNE
5 REM SYNTAX ERROR SE FUNZIONE MAL DEFINITA
7 REM CARICA ROUTINE CHE TRACCIA I PUNTI
9 BLOAD "RR8563",D0,U8,B15,P2816
11 REM INCORPORA LA FUNZIONE NEL PROGRAMMA
13 INPUT "F(X) ";F$
15 PRINT CHR$(147)CHR$(17)CHR$(17)CHR$(17);
17 PRINT CHR$(144)"23 DEF FNY(X)=" F$
19 PRINT "GOTO 23"CHR$(19)
21 POKE 208,2:POKE 842,13:POKE 843,13:END
23 DEF FNY(X)=X*SIN(X)
25 PRINT CHR$(154)CHR$(147)
27 INPUT "DOMINIO DA ";ID
29 INPUT "A ";FD
31 PRINT:PRINT "DISEGNO L'ASSE DELLE X ?"
33 GETKEY A$:IF A$<>"S" AND A$<>"N" THEN 33
35 IF A$="N" THEN AX=0:ELSE AX=1
37 PRINT
39 PRINT "STO CALCOLANDO MASSIMO E MINIMO"
41 FAST:IR=0:FR=0:AD=FD-ID
43 FOR X=0 TO 639
45 I=ID+X*AD/640
47 Y=FNY(I)
49 IF IR>Y THEN IR=Y
51 IF FR<Y THEN FR=Y
53 NEXT X
55 AR=FR-IR
57 ZR=ABS(IR/AR*199)
59 REM PASSA IN GRAFICA MONOCROMATICA
61 SYS 2816,25,128
62 GOSUB 121:REM AZZERA PAGINE GRAFICHE
63 REM PREPARA INDIRIZZO USR
65 POKE 4633,17:POKE 4634,11
67 REM DISEGNA IN GRAFICA
69 FOR X=0 TO 639
71 I=ID+X*AD/640
73 Y=FNY(I)*199/AR+ZR
75 GOSUB 91
77 IF AX=1 THEN Y=ZR:GOSUB 91
79 NEXT X
81 GETKEY A$:REM ATTESA TASTO
83 GOSUB 121:REM AZZERA PAGINE GRAFICHE
85 BANK 15:SYS 52750,0:REM RIPRISTINA SITUAZIONE
```

```
87 SYS 2816,25,64:REM RIPRISTINA TESTO
89 PRINT"□":STOP
91 REM ROUTINE CHE TRACCIA UN PUNTO
93 REM DI COORDINATE X E Y
95 Y=199-Y:K=INT(Y)*80+INT(X/8)
97 DX=X AND 7
99 P=2↑(7-DX)
101 LO=K AND 255
103 HI=INT(K/256)
105 REM LEGGE IL BYTE
107 SYS 2816,18,HI:SYS 2816,19,LO
109 A=USR(31)
111 A=A OR P
113 REM SCRIVE IL BYTE AGGIORNATO
115 SYS 2816,18,HI:SYS 2816,19,LO
117 SYS 2816,31,A
119 RETURN
121 REM ROUTINE AZZERAMENTO MEMORIA
123 REM AZZERA LA PAGINA GRAFICA
125 SYS 2816,18,0:SYS 2816,19,0
127 FOR K=0 TO 63
129 SYS 2816,31,0:SYS 2816,30,0
131 NEXT K:RETURN
```

Vediamo il dettaglio delle linee da 59 in poi:
– 59/61, si passa in grafica ponendo 128 nel registro 25;
– 62, si richiama la routine per azzerare la memoria;
– 63/65, si pone nei byte 4633 e 4634 l'indirizzo della parte di routine RR8563 che viene utilizzata per leggere la memoria con USR;
– 67/79, facendo variare X da 0 a 639 si calcola Y, utilizzando la funzione definita alla linea 23 e i parametri calcolati durante la ricerca del massimo e del minimo della funzione nell'intervallo. Dopo aver calcolato le coordinate di un punto, si utilizza la routine che inizia alla linea 91 per tracciarlo. Se richiesta l'asse X essa viene tracciata;
– 81/89, parte finale del programma, che viene eseguita quando, terminato il grafico, si preme un tasto. Viene ripristinata la situazione del video testo azzerando la memoria, richiamando la routine del sistema a 52750 e sistemando il registro 25;
– 91/119, routine che disegna un punto applicando le formule prima esposte per calcolare l'indirizzo e la posizione del bit. La routine RR8563 viene utilizzata prima per leggere il contenuto del byte di indirizzo K e poi per riscriverlo aggiornato;
– 121/131, routine di azzeramento della memoria in blocchi di 256 byte per volta.

2.8 I JOYSTICK

Per leggere lo stato dei Joystick si puó utilizzare la funzione JOY(N) del BASIC (con N=1 o 2) o lavorare direttamente sui registri dei chip 6526. L'istruzione JOY puó essere utilizzata sia per trasferire lo stato dei joystick in una variabile, che per analizzarlo direttamente, cioè:
A = JOY(1) oppure IF JOY(1) = ...

nel primo caso viene poi analizzata la variabile A.
I valori forniti dalla funzione sono 9 più 9; essi corrispondono rispettivamente agli 8 movimenti possibili più lo stato di riposo, ottenuti con il solo movimento della leva, e agli 8 movimenti possibili più lo stato di riposo, ottenuti premendo contemporaneramente alla leva anche il bottone del fuoco. Nella figura 2.4 è riportato lo schema dei valori possibili, scritti nelle corrispondenti posizioni.

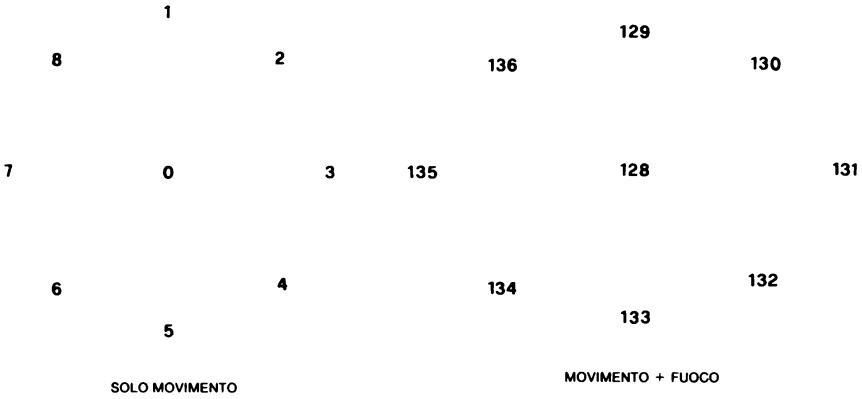


Fig. 2.4 Lettura della situazione dei Joystick

I joystick possono essere utilizzati per produrre movimento leggendo il loro stato e traducendolo in spostamenti di figure sul video.

Abbiamo preparato il programma ES29 che lavora in modo testo e, per effetto dell'azione sul joystick 2, produce lo spostamento di un quadratino, ottenuto con il carattere spazio in campo inverso, nelle possibili direzioni. Il quadratino è nero e diventa rosso se si preme anche il bottone del fuoco. Inizialmente esso viene posto nella posizione 14,14 di coordinate testo, e viene spostato, dopo averlo cancellato; viene utilizzato un ciclo di attesa per consentire di vedere bene il movimento. Dopo ogni spostamento il programma ricomincia riportando il quadratino nella posizione iniziale.
Le variabili utilizzate sono:
- T, controllo ciclo attesa;
- B$, spostamento verso il basso partendo dall'angolo in alto a sinistra;
- D$, spostamento verso destra;
- V$(8), stringhe di spostamento verticale per gli 8 possibili spostamenti;
- H$(8), stringhe di spostamento orizzontale per gli 8 possibili spostamenti;
- S$, spazio in campo inverso;
- N$, spazio;
- J, lettura joystick;
- K, controllo cicli.

La situazione delle coordinate delle posizioni possibili è la seguente:

13,13 13,14 13,15

14,13 14,14 14,15
15,13 15,14 15,15.

Il programma prepara inizialmente le stringhe di caratteri di controllo per gli spostamenti, poi posiziona il quadratino in 14,14 e interroga lo stato del joystick 2. In base al valore letto determina il movimento o il movimento più il fuoco.

```
1 REM ES29
3 REM STRINGHE PER MOVIMENTO
5 T=200:REM CICLO DI ATTESA
7 B$= "[illegible]"
9 D$= "[illegible]"
11 DIM V$(8),H$(8):REM STRINGHE MOVIMENTO
13 V$(1)=LEFT$(B$,14):H$(1)=LEFT$(D$,14)
15 V$(2)=V$(1):H$(2)=LEFT$(D$,15)
17 V$(3)=LEFT$(B$,15):H$(3)=H$(2)
19 V$(4)=LEFT$(B$,16):H$(4)=H$(3)
21 V$(5)=V$(4):H$(5)=H$(1)
23 V$(6)=V$(5):H$(6)=LEFT$(D$,13)
25 V$(7)=V$(3):H$(7)=H$(6)
27 V$(8)=V$(1):H$(8)=H$(6)
29 REM SPAZIO INVERSO E DIRETTO
31 S$="[illegible]":N$=" "
33 REM QUADRATINO A RIGA 14 COLONNA 14
35 PRINT "[illegible]";V$(3);H$(1);"[illegible]";S$
37 REM LETTURA JOYSTICK 2
39 J=JOY(2):IF J=0 THEN 39
41 FOR K=1 TO T:NEXT K :REM CICLO ATTESA
43 PRINT V$(3);H$(1);N$;:REM CANCELLA
45 IF J=128 THEN 67
47 IF J>128 THEN 59
49 REM SOLO MOVIMENTO
51 IF J<1 OR J>8 THEN STOP
53 PRINT V$(J);H$(J);S$;:REM RISCRIVE
55 REM CICLO DI ATTESA, POI RICOMINCIA
57 FOR K=1 TO T:NEXT K:J=0:GOTO 33
59 REM MOVIMENTO + FUOCO
61 J=J-128
63 IF J<1 OR J>8 THEN STOP
65 PRINT V$(J);H$(J);"[illegible]";S$;:GOTO 55
67 REM SOLO FUOCO
69 PRINT V$(3);H$(1);"[illegible]";S$;:GOTO 55
```

Il programma ES30 lavora invece in modo grafico e ci permette di mostrare l'utilizzo delle istruzioni GSHAPE e SSHAPE. Il programma lavora con il joistick 1.
All'inizio prepariamo un piccolo rombo disegnandolo con l'istruzione DRAW, e lo memorizziamo nella stringa S$ utilizzando l'istruzione SSHAPE.
Partiamo dalla posizione 154,99 in coordinate grafiche. Il programma sposta con

continuità il rombo sul video, colorandolo se si è premuto anche il bottone del fuoco. Gli spostamenti programmati sono relativi alla posizione attuale del rombo, e sono i seguenti, in coordinate relative:

-10,-10	0,-10	10,-10
-10, 0		10, 0
-10, 10	0, 10	10, 10.

Anche in questo caso il rombo va prima cancellato e poi spostato; si utilizza un ciclo di attesa per rendere ben visibile il movimento. Per disegnare il rombo utilizziamo l'istruzione GSHAPE, e per cancellarlo eseguiamo ancora la stessa istruzione, ma con il parametro *modalità* che esegue l'operazione XOR della figura con la situazione precedente e quindi la cancella.
Le variabili utilizzate sono:
- T, ciclo attesa;
- X,Y, posizioni iniziali;
- X1,Y1, coordinate attuali;
- IX(8),IY(8), vettori per incrementi coordinate X1 e Y1;
- J, lettura joystick;
- K, controllo cicli;
- S$, disegno del rombo;
- C, numero sorgente colore.

```
1 REM ES30
3 REM ATTIVA MODO GRAFICO
5 GRAPHIC 1,1
7 T=150:REM CICLO ATTESA
9 COLOR 1,1:REM COLORE PRIMO PIANO
11 REM PREPARA SHAPE, PICCOLO ROMBO
13 X=159:Y=99
15 DRAW 1,X,Y TO X+5,Y+5 TO X,Y+10 TO X-5,Y+5 TO X,Y
17 SSHAPE S$,X-5,Y,X+5,Y+10
19 X1=X-5:Y1=Y
21 DIM IX(8),IY(8):REM VETTORI MOVIMENTO
23 REM INCREMENTI COORDINATE
25 IX(1)=0:IY(1)=-10
27 IX(2)=10:IY(2)=-10
29 IX(3)=10:IY(3)=0
31 IX(4)=10:IY(4)=10
33 IX(5)=0:IY(5)=10
35 IX(6)=-10:IY(7)=10
37 IX(7)=-10:IY(7)=0
39 IX(8)=-10:IY(8)=-10
41 REM INTERROGA JOYSTICK 1
43 J=JOY(1)
45 IF J=0 THEN 43
47 FOR K=1 TO T:NEXT K:REM CICLO ATTESA
49 GSHAPE S$,X1,Y1:REM RIDISEGNA
51 GSHAPE S$,X1,Y1,4:REM CANCELLA
```

```
53 IF J=128 THEN 79
55 IF J>128 THEN 71
57 C=0:REM SORGENTE COLORE
59 IF J<1 OR J>8 THEN STOP
61 X1=X1+IX(J):Y1=Y1+IY(J):REM MODIFICA COORDINATE
63 GSHAPE S$,X1,Y1:REM DISEGNA
65 PAINT C,X1+5,Y1+5,1:REM COLORA
67 FOR K=1 TO T:NEXT K:REM CICLO ATTESA
69 J=0:GOTO 43
71 REM FUOCO + MOVIMENTO
73 J=J-128
75 C=1:REM SORGENTE COLORE
77 GOTO 59
79 C=1:REM SORGENTE COLORE
81 GOTO 63
```

2.9 IL MOVIMENTO E L'ANIMAZIONE DELLE FIGURE

Il movimento di una figura consiste nel farla scivolare da una posizione ad un altra, l'animazione invece consiste nella simulazione di un'azione reale, come camminare muovendo le gambe.

Il COMMODORE 128 ci consente di utilizzare gli *sprite*, per muovere e animare le figure, in modo molto più semplice rispetto al COMMODORE 64; infatti nel BASIC 7.0 sono disponibili parecchie istruzioni per la gestione degli sprite, e non risulta necessario lavorare direttamente sui registri del chip 8564. Anche in questo caso sono gestibili contemporaneamente 8 sprite.
Gli sprite sono delle figure comprese in un rettangolo avente le dimensioni di 24 per 21 punti; rapportandolo alla grandezza di un carattere in modo testo, uno sprite corrisponde a circa 9 caratteri. Una figura si compone per punti utilizzando un rettangolo di carta a quadretti 24 per 21; il sistema utilizza per memorizzarla 63 byte. Esiste un'area privilegiata dove il sistema memorizza gli sprite; essa si trova dal byte 3584 ($0E00) al byte 4095 ($0FFF) e possiamo considerarla come l'insieme degli 8 buffer degli sprite. Gli sprite sono memorizzati a intervalli di 64 byte; esistono 8 puntatori agli sprite, situati, quando si lavora in modo testo con la mappa video a 1024, da 2040 a 2047, e quando si lavora in grafica, con la mappa ad alta risoluzione a 8192, da 8184 a 8191. Tali puntatori contengono un numero che moltiplicato per 64 dà l'indirizzo di inizio della descrizione dello sprite corrisponente; dopo un reset il valore dei puntatori va da 56 a 63.
Il sistema gestisce gli sprite in modo indipendente dal resto; si puó mettere in movimento uno sprite e poi svolgere altri lavori. Per fare sparire gli sprite si puó utilizzare RUN/STOP-RESTORE.
Ogni sprite puó avere le seguenti caratteristiche:
– essere di un colore, quello di primo piano;
– essere di tre colori, primo piano, multicolore 1 e multicolore 2;
– avere priorità più alta dello sfondo e quindi passare sopra gli oggetti presenti sul video;
– avere priorità più bassa dello sfondo e quindi passare dietro gli oggetti presenti sul video.
Inoltre, uno sprite puó essere espanso sia in direzione orizzontale che verticale,

singolarmente o contemporaneamente. Da uno sprite se ne possono quindi ricavare 4 di dimensioni diverse.
Nelle istruzioni BASIC il numero distintivo dello sprite va da 1 a 8, anche se internamente i numeri sono, come sempre, calcolati da 0 a 7; succede la stessa cosa con i numeri dei colori. Gli sprite hanno una priorità tra loro, cioè se si incontrano passa davanti quello che ha priorità maggiore e che corrisponde al numero minore; tale comportamento è schematizzato nella figura 2.5.

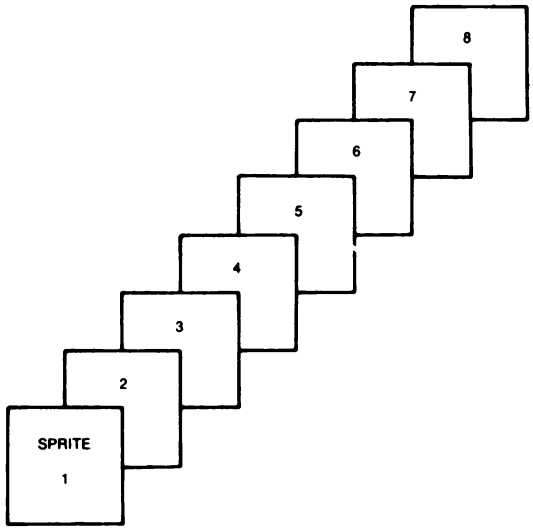


Fig. 2.5 Priorità degli sprite

Gli sprite possono essere utilizzati sia lavorando in modo testo a 40 colonne che in grafica a 40 colonne, normale e multicolore; il fatto di utilizzare sprite in tre colori (multicolori) non richiede di lavorare in grafica multicolore.

Per preparare gli sprite si possono utilizzare quattro metodi:
- 1) utilizzare l'istruzione SPRDEF, che consente di disegnare in modo semplice e veloce gli sprite sul video;
- 2) disegnare, anche utilizzando i comandi grafici, sul video uno sprite, facendo attenzione alle dimensioni;
- 3) disegnare sulla carta lo sprite, dividere in gruppi di 8 quadratini partendo dall'alto e calcolare i valori per i byte corrispondenti;
- 4) disegnare lo sprite come sequenza di stringhe in frasi DATA nel programma e far calcolare al programma i valori corrispondenti per i byte.

Il metodo 1 è il più semplice, infatti l'istruzione SPRDEF presenta sul video ben ingrandita la sagoma del rettangolo da utilizzare per disegnare, con un cursore spostabile, sotto forma di '+', che viene posizionato utilizzando i tasti di movimento del cursore. Per disegnare si usano i tasti numerici da 1 a 4: 1 dà il colore dello

sfondo, 2 il colore di primo piano, 3 il colore multicolor 1 e 4 il colore multicolor 2 (questi ultimi due vanno usati solo se si è scelta l'opzione M per multicolore). Oltre che vedere lo sprite ingrandito mentre si disegna, se ne vede a destra l'immagine come apparirà; in conseguenza si controlla bene come procede il lavoro. Finito di disegnare una figura, premendo SHIFT-RETURN, il disegno viene mandato nell'area che gli compete in base al numero di sprite selezionato. Sono disponibili molti comandi, ed è anche possibile copiare uno sprite già memorizzato con altro numero. Quando si esce dall'istruzione SPRDEF gli sprite sono memorizzati nella loro area; a questo punto conviene trasferirli su disco come file binario con BSAVE, assegnando un nome al file. Se non si vuole memorizzarli su disco, si puó passare in MONITOR e utilizzare il comando S, che consente di salvare un file binario anche su cassetta.

Il metodo 2 è spiegato con un esempio nel manuale; viene preparato il disegno in modo grafico e poi memorizzato con l'istruzione SSHAPE in una stringa, dalla quale con l'istruzione SPRSAV lo sprite viene portato nella sua area. Anche in questo caso, come pure negli altri, conviene trasferire gli sprite su disco o cassetta in un file binario, prelevandoli dalla loro area.

Il metodo 3 consiste nel considerare i quadratini anneriti come bit 1 e quelli non anneriti come bit 0 di un byte, dopo aver diviso ogni linea del disegno in 3 (3*8=24), e nel calcolare il valore di ogni byte. I 63 numeri che si ottengono devono essere memorizzati in sequenza nei primi 63 byte della zona di memoria dedicata allo sprite.

Il metodo 4 è stato da noi utilizzato nei programmi riportati nel libro dedicato alla grafica del COMMODORE 64, e ad esso rimandiamo il lettore.

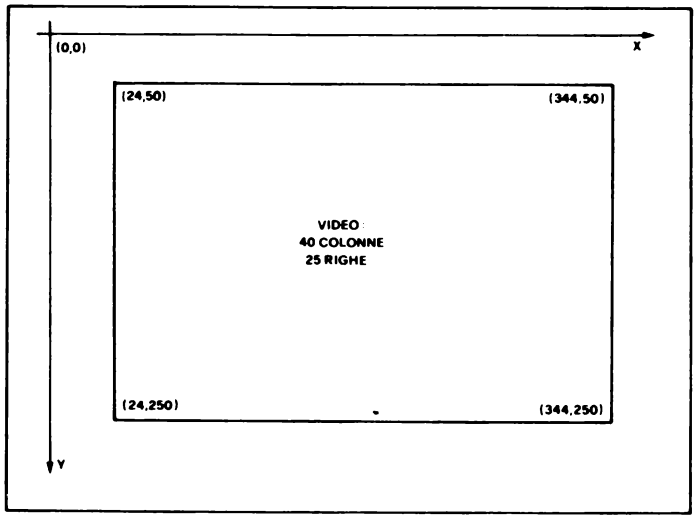


Fig. 2.6 Quadro video visibile (40 x 25) per gli sprite

Dopo avere memorizzato gli sprite nella loro zona di memoria per poterli vedere è necessario:
- posizionarli sul video;
- renderli attivi;

farli muovere, o animarli, secondo l'effetto che si desidera ottenere.

La zona video *visibile*, cioè quella nella quale è visibile uno sprite, ha coordinate diverse da quelle grafiche o di testo; l'asse X si trova in alto orientato verso destra, fuori dalla zona visibile, e l'asse Y si trova a sinistra orientato verso il basso, fuori dalla zona visibile. Riportiamo nelle figure 2.6 e 2.7 i limiti delle coordinate per vedere uno sprite per lo schermo 40 per 25 e per lo schermo 38 per 24.

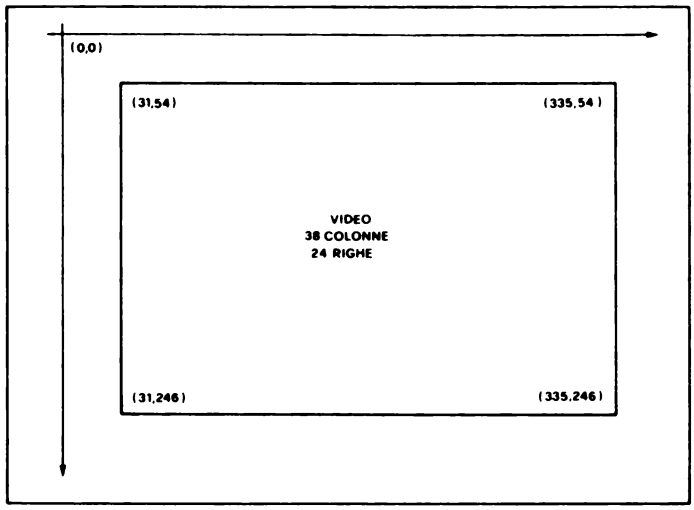


Fig. 2.7 Quadro video visibile (38 x 24) per gli sprite

Gli sprite possono essere attivi anche fuori dalla zona visibile dello schermo; le coordinate di schermo per gli sprite sono *circolari*, cioè uno sprite in movimento se esce a sinistra del video, dopo un po' ricompare da destra, se non viene disattivato.
Per stabilire le coordinate da assegnare a uno sprite perché sia completamente visibile bisogna tener conto che esse si riferiscono all'angolo in alto a sinistra del rettangolo che lo contiene.

Abbiamo preparato alcuni programmi esempio che utilizzano gli sprite, e, a questo scopo, abbiamo preparato e memorizzato su disco 3 file binari contenenti alcune figure. Riportiamo nelle figure 2.8 e 2.9 gli sprite disegnati. Quelli della figura 2.8 sono ad alta risoluzione in un solo colore e li abbiamo disegnati annerendo i quadratini.

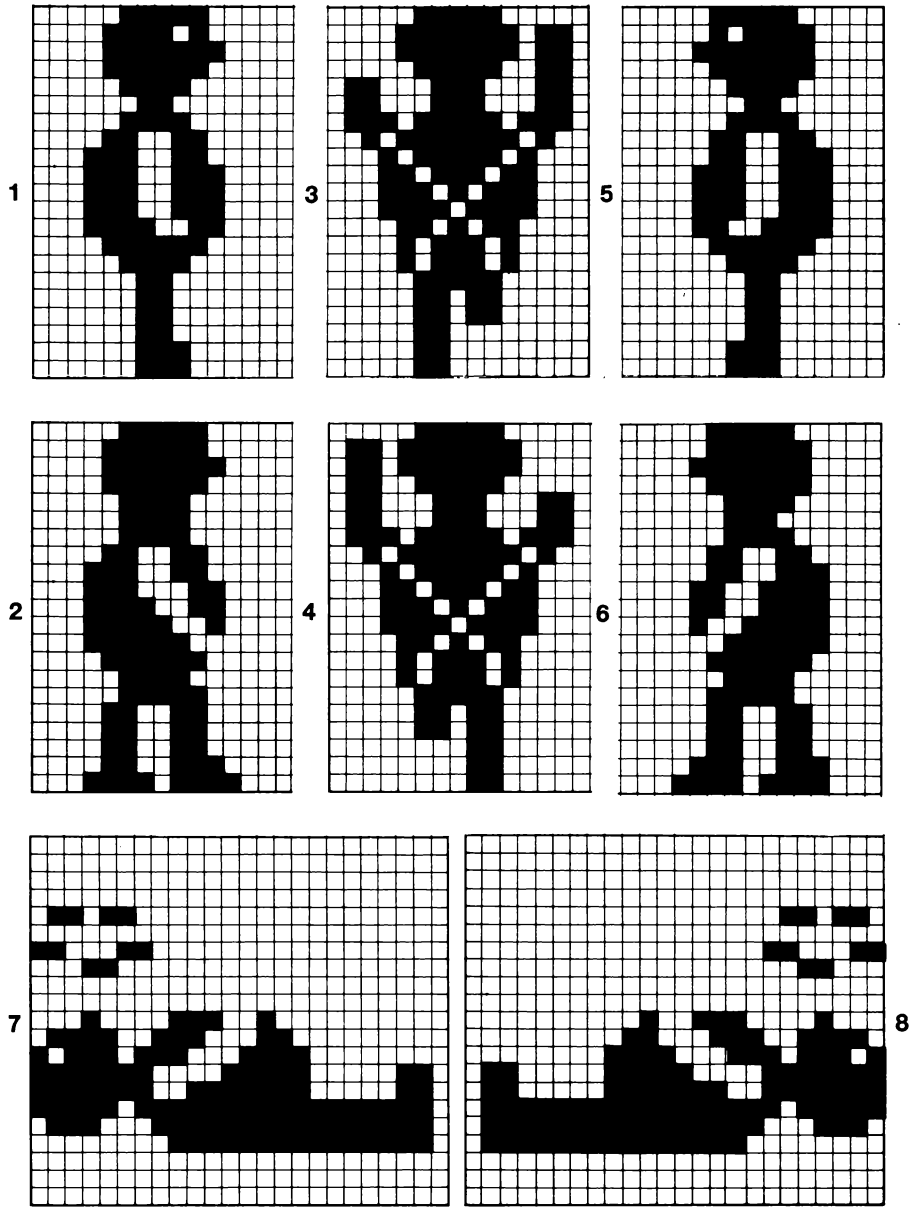


Fig. 2.8 Sprite ad alta risoluzione per file FSPRA e FSPRB

Abbiamo utilizzato il disegno 1 e il disegno 2 per creare un file binario FSPRA nel quale abbiamo memorizzato come sprite 1, 2, 3 e 4 il disegno 1, cioè l'omino a gambe unite, e come sprite 5,6,7 e 8 il disegno 2, cioè l'omino in posizione di marcia.
Abbiamo utilizzato gli 8 disegni in sequenza per creare il file binario FSPRB, con 8 sprite diversi.

Nella figura 2.9, che riguarda sprite multicolori, i disegni sono larghi la metà, infatti ogni quadratino rappresenta un numero di due bit, e darà luogo a due punti. Noi abbiamo utilizzato i numeri da 2 a 4 per indicare i tasti da premere con l'istruzione SPRDEF; è evidente che verranno memorizzati i bit: 00 per lo sfondo e 01, 10, 11 per i punti colorati, cioè il numero usato da noi nel disegno meno uno. Nell'istruzione SPRDEF con l'opzione M il cursore viene rappresentato da due caratteri '+' vicini.

1

	2	2	
	2	4	4
	2	4	4
	4	4	
	3	4	
3	4	3	
3	4	3	3
3	4	3	3
3	4	3	3
3	4	3	3
2	4	2	2
2	2	2	2
	2	2	2
	4	4	
	4	4	
	3	3	
	3	3	2
	2	2	2

2

					2	2					
					2	4	4				
					2	4	4				
					4	4					
			3	3	4	4	3	3			
		4	4	3	3	3	3	4			
	4	4		3	3	3	3	4			
		4	4	3	3	3	3	4			
			4	4	3	3	3	4	4	4	4
				3	3	3	3				
				2	2	2	2				
				2	2	2	2	2			
			4	4	4	2	2	4	4		
2	3	3	4	4				4	4	4	
2	3	3	4						4	4	
2									3	3	
									3	3	
									2	2	2

3

		2	2			
		2	4	4		
		2	4	4		
		4	4			
	3	3	4			
	4	3	3	3		
	4	3	3	3		
	4	3	3	3		
	4	4	4	4	4	4
	3	3	3	3		
	2	2	2	2		
	2	2	2	2		
		2	2	2		
2	3	4	4	4		
2	3	4	4	4		
2		3	3			
		3	3			
		2	2	2		

4

					2	2				
					2	4	4			
					2	4	4			
					4	4				
				3	3	3				
			4	3	3	4	3			
		4	4	3	3	4	3			
			4	3	3	4	3			
				3	3	4	4	4	4	
				3	3	3	3			
				2	2	2	2			
				2	2	2	2	2		
			4	4	2	2	2	2		
		4	4	4			4	4	4	
	3	4	4					3	4	4
3	3	3					2	3	3	
2	3						2	3		
	2	2						2		

5

						2	2				
						2	4	4			
						2	4	4			
						4	4				
				3	3	3	3	3	3		
			4	4	3	3	3	3	4		
		4	4		3	3	3	3	4		
			4	4	3	3	3	3	4		
				4	3	3	3	3	4	4	4	4
					3	3	3	3			
					2	2	2	2			
					2	2	2	2	2		
				4	4	2		2	4	4	
2	3	3		4	4				4	4	4	
2	3	3		4						4	4	
2										3	3	
										3	3	
										2	2	2

6

		2	2			
		2	4	4		
		2	4	4		
		4	4			
	3	3	3			
	3	3	4	3		
	3	3	4	3		
	3	3	4	3		
	3	3	4	4	4	4
	3	3	3	3		
	2	2	2	2		
	2	2	2	2		
		2	2	2		
2	3	4	4	4		
2	3	4	4	4		
2		3	3			
		3	3			
		2	2	2		

7

					2	2				
					2	4	4			
					2	4	4			
					4	4				
				3	3	3				
			4	4	3	3	3			
		4	4	3	3	3	3			
			4	4	3	3	3			
				4	4	3	3	4	4	
				3	3	3	3			
				2	2	2	2			
				2	2	2	2			
			4	4	2	2	4	4		
		4	4	4			4	4	4	
	3	4	3					3	4	4
3	3	3					2	3	3	
2	3						2	3		
	2	2						2		

Fig. 2.9 Sprite multicolori per file FSPRC

Il file degli sprite multicolori è stato chiamato FSPRC e contiene 7 sprite diversi.

Nel programma ES31 vogliamo simulare il cammino di 4 omini ottenuti da un disegno utilizzato sia in modo normale che con le tre possibili espansioni. Per simulare il cammino dobbiamo alternare le figure a gambe unite con quelle a gambe aperte e modificare le coordinate delle figure a opportuni intervalli di tempo. Utilizzando gli sprite non è necessario cancellare la figura nella vecchia posizione prima di metterla nella nuova, basta disattivare lo sprite nella vecchia posizione e attivare l'altro sprite nella nuova posizione alternativamente. Noi utilizziamo il file FSPRA, nel quale i primi 4 sprite rappresentano omini a gambe chiuse e gli altri 4 omini a gambe aperte. Il programma esegue i seguenti passi:
– predispone i colori e le variabili di lavoro;
– carica in memoria il file che contiene gli sprite;
– inizia un ciclo controllato dal valore della coordinata X, nella variabile K, che inizia con 24 (prima coordinata *visibile*) e termina con 1200, con passo S, posto da noi uguale a 4. Questo significa che le figure girano, uscendo dal video e poi tornano visibili, arrestandosi ad un certo punto;
– disattiva gli sprite 5, 6, 7 e 8;
– attiva gli sprite 1, 2, 3, 4 utilizzando i parametri necessari per espandere gli ultimi tre;
– posiziona in base al valore di K, e di una coordinata Y opportuna, gli omini;
– dopo un ciclo di attesa controllato dal valore di A, posto inizialmente a 120, disattiva gli sprite 1, 2, 3 e 4;
– attiva gli sprite 5, 6, 7, 8, assegnandogli gli stessi attributi di prima;
– assegna nuove coordinate ai 4 sprite, incrementando il vecchio valore di K con P, posto inizialmente da noi uguale a 4, cioè uguale allo STEP S del ciclo FOR;
– crea un secondo ciclo di attesa e ripercorre il ciclo esterno.
Le variabili utilizzate, oltre a J che controlla i cicli di attesa, sono S, A e P, che abbiamo posto all'inizio per consentire di fare delle prove modificandone rapidamente i valori.

```
1 REM ES31
3 REM MOVIMENTO OMINI
5 REM COLORE SFONDO E BORDO
7 COLOR 0,2:COLOR 4,2
9 REM STEP, INCREMENTO PASSO, RITARDO
11 S=4:P=4:A=120
13 REM CARICA FILE FSPRA CON 8 SPRITE
15 BLOAD "FSPRA",B0
17 REM PULISCE IL VIDEO
19 PRINT"]"
21 REM CICLO DI MOVIMENTO SPRITE
23 FOR K=24 TO 1200 STEP S
25 REM DISATTIVA SPRITE 5, 6, 7, 8
27 SPRITE 5,0:SPRITE 6,0
29 SPRITE 7,0:SPRITE 8,0
31 REM ATTIVA SPRITE 1, 2, 3, 4
33 SPRITE 1,1,3:SPRITE 2,1,7,,0,1
35 SPRITE 3,1,1,,1,0:SPRITE 4,1,5,,1,1
37 REM POSIZIONA OMINI GAMBE CHIUSE
```

```
39 MOVSPR 1,K,50:MOVSPR 2,K,90
41 MOVSPR 3,K,148:MOVSPR 4,K,190
43 REM CICLO ATTESA
45 FOR J=1 TO A:NEXT J
47 REM DISATTIVA OMINO GAMBE UNITE
49 SPRITE 1,0:SPRITE 2,0
51 SPRITE 3,0:SPRITE 4,0
53 REM ATTIVA OMINO GAMBE APERTE
55 SPRITE 5,1,3
57 SPRITE 5,1,3:SPRITE 6,1,7,,0,1
59 SPRITE 7,1,1,,1,0
61 SPRITE 7,1,1,,1,0:SPRITE 8,1,5,,1,1
63 REM MUOVE OMINO GAMBE APERTE
65 MOVSPR 5,K+P,50:MOVSPR 6,K+P,90
67 MOVSPR 7,K+P,148:MOVSPR 8,K+P,190
69 REM CICLO ATTESA
71 FOR J=1 TO A:NEXT J
73 NEXT K
```

Abbiamo preparato il programma ES32 per mostrare come si puó produrre il movimento utilizzando l'istruzione MOVSPR con il parametro 'angolo # velocità'. Utilizziamo ancora il file FSPRA degli sprite, ma ci serviamo solo delle figure a gambe aperte, ottenendo 4 figure diverse con le possibili espansioni. Abbiamo posizionato degli ostacoli sul video stampando delle stringhe in campo inverso. Abbiamo assegnato agli sprite 5 e 6 la priorità 0, per farli apparire davanti agli oggetti, e agli sprite 7 e 8 la priorirà 1, per farli aparire dietro. Disattiviamo gli sprite da 1 a 4, che non usiamo; poi attiviamo gli altri sistemando i parametri e li posizioniamo a sinistra sul video. Attiviamo il movimento con quattro istruzioni MOVSPR, dando ad ogni sprite angolo e velocità diversa. Le figure si spostano continuamente sul video, si scontrano tra loro e si scontrano con gli ostacoli; quando si scontrano due sprite si vede passare davanti quello con numero minore, che ha la priorità più alta.

```
1 REM ES32
3 REM MOVIMENTO OMINI
5 REM COLORE SFONDO E BORDO E CARATTERI
7 COLOR 0,2:COLOR 4,2:COLOR 5,10
9 REM CARICA FILE FSPRA CON 8 SPRITE
11 BLOAD "FSPRA",B0
13 REM PULISCE IL VIDEO
15 PRINT"▐"
17 REM PONE OSTACOLI SUL VIDEO
19 O$="▐      ▌"
21 PRINT "▐▌▌▌▐▐▐▐" O$
23 PRINT "▐▌▌▌" O$
25 PRINT "▐▌▌▌" O$
27 PRINT "▐▐▐▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌" O$
29 PRINT "▐▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌" O$
```

```
31 PRINT "▮▮▮
```

```
23 REM DISATTIVA TUTTI GLI SPRITE
25 FOR J=1 TO 8:SPRITE J,0: NEXT J
27 REM PULISCE IL VIDEO
29 PRINT"□"
31 REM ATTIVA I PRIMI 4 SPRITE
33 SPRITE 1,1,3:SPRITE 2,1,7,,0,1
35 SPRITE 3,1,1,,1,0:SPRITE 4,1,5,,1,1
37 REM CARICA SPRITE 1, 2, 3, 4 CON S1$
39 FOR J=1 TO 4:SPRSAV S1$,J: NEXT J
41 REM POSIZIONA OMINI GAMBE CHIUSE
43 MOVSPR 1,K,50:MOVSPR 2,K,90
45 MOVSPR 3,K,148:MOVSPR 4,K,190
47 REM AVVIA MOVIMENTO ORIZZONTALE OMINI
49 FOR J=1 TO 4:MOVSPR J,90#1: NEXT J
51 REM CICLO ALTERNANZA OMINI
53 FOR J=1 TO A:NEXT J:REM CICLO ATTESA
55 REM CARICA OMINO GAMBE APERTE
57 FOR J=1 TO 4:SPRSAV S2$,J:NEXT J
59 FOR J=1 TO A:NEXT J:REM CICLO ATTESA
61 REM CARICA OMINO GAMBE CHIUSE
63 FOR J=1 TO 4: SPRSAV S1$,J: NEXT J
65 GOTO 53
```

Questo metodo, di memorizzare le figure in stringhe e di caricarle nella memoria propria degli sprite quando servono, consente di utilizzare un numero di figure diverse superiore a 8. Si possono preparare diversi file di figure, caricarli in memoria in stringhe, e utilizzarli. Nel programma ES33 gli sprite restano sempre attivi, ma viene cambiata la loro natura mentre si muovono. Si puó provare a modificare la lunghezza del ciclo d'attesa e la velocità per trovare la situazione ottimale.

Con il programma ES34 ci siamo proposti di risolvere il problema di far muovere due omini in direzione contraria a due diversi livelli del video, dopo aver circondato il video con una cornice. Quando gli omini urtano contro la cornice, invece di proseguire dietro il video e riapparire dall'altra parte, li facciamo voltare e tornare indietro. Utilizziamo l'istruzione COLLISION, che si presenta un po' difficile da usare, infatti essa sfrutta le interruzioni prioritarie e il BASIC risulta troppo lento per gestire bene le interruzioni. Inoltre l'istruzione COLLISION fa scattare la routine predisposta, ma non distingue tra loro le possibili collisioni; dopo aver disattivato COLLISION si puó utilizzare l'istruzione BUMP per sapere quali sprite sono entrati in collisione. Noi, per semplificare le cose, quando scatta la routine di servizio di COLLISION, disattiviamo la segnalazione di collisioni e lavoriamo sui due omini per cambiare la direzione delle figure, ma utilizziamo una sola figura, quella con le gambe aperte; alla fine della routine, prima di RETURN, riattiviamo la segnalazione di collisioni.
Il programma ES34 lavora in modo testo e costruisce utilizzando lo spazio inverso, codice ASCII 160, una cornice intorno al video. Inizialmente viene letto in memoria il file FSPRB contenente gli sprite; di essi vengono utilizzati il secondo e il sesto che rappresentano due figure che guardano in senso opposto. Le due figure vengono memorizzate nelle stringhe D$ e S$ e da queste prelevate e poste alternativamente nei buffer degli sprite 1 e 2. Nelle linee da 43 a 51 vengono attivati

e posizionati i due sprite. Poi viene attivata la routine per COLLISION, con parametro 2, per scontro sprite/oggetti, che inizia alla linea 67. Dopo aver attivato COLLISION il programma continua; quando scatta un'interruzione per collisione viene eseguita la routine che inizia alla linea 67. Alla conclusione della routine il controllo torna al programma, all'istruzione successiva a quella al cui termine è scattato il servizio dell'interruzione. Alle linee 59 e 61 vengono messi in moto i due omini con velocità 1, la più bassa, e angoli 90 e 270 per farli procedere orizzontalmente in senso contrario e viene posta a zero la variabile S, flag che controlla le operazioni svolte nella routine di servizio. L'istruzione della linea 63 crea un ciclo infinito. Per interrompere il programma si deve premere RUN/STOP-RESTORE.
Nella routine di servizio, dopo aver disattivato l'interruzione, a seconda del valore del flag S si eseguono i seguenti passi:
- azzeramento delle velocità dei due sprite per fermarli;
- breve ciclo di attesa;
- scambio delle figure nei due buffer;
- riposizionamento dei due sprite;
- riavvio in direzione contraria;
- cambio valore del flag S.

```
1 REM ES34
3 REM OMINI CHE CORRONO
5 REM CARICA FILE FSPRB CON SPRITE
7 BLOAD"FSPRB",B0
9 REM SISTEMA COLORI
11 COLOR 0,1:COLOR 4,1:COLOR 5,2:COLOR 1,8
13 PRINT"□":CLR:REM AZZERA VARIABILI
15 REM TRACCIA UNA CORNICE
17 FOR K=1024 TO 1063:POKE K,160:BANK 15
19 POKE 54272+K,PEEK(54272+K) AND 240 OR 7: NEXT K
21 FOR K=1984 TO 2023: POKE K, 160:BANK 15
23 POKE 54272+K,PEEK(54272+K) AND 240 OR 7:NEXT K
25 FOR K=1024 TO 1984 STEP 40: POKE K,160:BANK 15
27 POKE 54272+K,PEEK(54272+K) AND 240 OR 7:NEXT K
29 FOR K=1063 TO 2023 STEP 40: POKE K, 160:BANK 15
31 POKE 54272+K,PEEK(54272+K) AND 240 OR 7:NEXT K
33 REM STRINGHE PER SPRITE D$ E S$
35 SPRSAV 2,D$:SPRSAV 6,S$
37 REM ATTIVA SPRITE 1 E 2
39 SPRSAV D$,1:REM OMINO A DESTRA IN 1
41 SPRSAV S$,2:REM OMINO A SINISTRA IN 2
43 SPRITE 1,1,3,1,1,1,0:REM ATTIVA 1 ROSSO
45 SPRITE 2,1,6,1,1,1,0:REM ATTIVA 2 BLU
47 REM POSIZIONA OMINI
49 MOVSPR 1,32,100:REM POSIZIONA 1
51 MOVSPR 2,304,150:REM POSIZIONA 2
53 COLLISION 2:REM RESETTA PRECEDENTI
55 COLLISION 2,67:REM SE COLLISIONE VA A 2000
57 REM INIZIA MOVIMENTO
59 S=0:MOVSPR 1,90#1
```

```
61 MOVSPR 2,270#1
63 GOTO 63
65 STOP
67 REM CONTROLLA COLLISIONE CON BORDO
69 COLLISION 2:REM INTERROMPE COLLISION
71 IF S=1 THEN 93
73 REM AZZERA VELOCITA'
75 MOVSPR 1,90#0:MOVSPR 2,270#0
77 FORJ=1TO100:NEXT J:REM ATTESA
79 REM CAMBIA VERSO OMINI
81 SPRSAV S$,1:SPRSAV D$,2
83 REM POSIZIONA OMINI
85 MOVSPR 2,32,150:MOVSPR 1,304,100
87 REM ASSEGNA VELOCITA' IN DIVERSA DIREZIONE
89 MOVSPR 2,90#1:MOVSPR 1,270#1
91 S=1:GOTO 111
93 REM AZZERA VELOCITA'
95 MOVSPR 2,90#0:MOVSPR 1,270#0
97 FORJ=1TO100:NEXT J:REM ATTESA
99 REM CAMBIA VERSO OMINI
101 SPRSAV D$,1:SPRSAV S$,2
103 REM POSIZIONA OMINI
105 MOVSPR 1,32,100:MOVSPR 2,304,150
107 REM ASSEGNA VELOCITA' IN DIVERSA DIREZIONE
109 MOVSPR 1,90#1:MOVSPR 2,270#1:S=0
111 COLLISION 2,67:RETURN
```

Si consiglia di provare a migliorare il programma facendo camminare gli omini, invece che scivolare.

Nel programma ES35 abbiamo utilizzato gli sprite multicolori creati nel file FSPRC per far correre un atleta. All'inizio vengono scelti i colori e inizializzate le variabili per controllare il ciclo di attesa e la velocità; poi viene caricato in memoria il file con i 7 sprite del corridore e questi vengono trasferiti in una matrice di stringhe con 7 elementi. Vengono inoltre disattivati tutti gli sprite, operazione eseguita per sicurezza, dato che ne possono essere rimasti attivi alcuni a causa di programmi eseguiti prima.
Viene attivato lo sprite 1, viene posizionato e messo in movimento; dopo il programma continua con un ciclo infinito che alterna le 7 figure nel buffer dello sprite 1, simulando la corsa. Il programma si interrompe con RUN/STOP-RESTORE.

```
1 REM ES35
3 REM CORRIDORE
5 REM COLORE SFONDO E BORDO
7 COLOR 0,1:COLOR 4,1
9 REM CICLO ATTESA SCAMBIO FIGURA E VELOCITA'
11 A=50:V=3
13 REM CARICA FILE FSPRC CON 8 SPRITE
```

```
15 BLOAD "FSPRC",B0
17 REM CARICA SPRITE IN MATRICE STRINGA
19 DIM S$(7):REM MATRICE PER SPRITE
21 FOR J=1 TO 7: SPRSAV J,S$(J):NEXT J
23 REM DISATTIVA TUTTI GLI SPRITE
25 FOR J=1 TO 7:SPRITE J,0: NEXT J
27 REM PULISCE IL VIDEO
29 PRINT"□"
31 REM ATTIVA PRIMO SPRITE
33 SPRITE 1,1,7,0,1,1,1
35 REM POSIZIONA PRIMO SPRITE
37 MOVSPR 1,32,150
39 REM AVVIA MOVIMENTO CORRIDORE
41 MOVSPR 1,90#V
43 REM CICLO ALTERNANZA FIGURE
45 FOR J=1 TO 7
47 SPRSAV S$(J),1:REM CARICA SPRITE 1
49 FOR I=1 TO A:NEXT I:REM CICLO ATTESA
51 NEXT J
53 GOTO 45
```

Abbiamo già accennato al fatto che nei byte di indirizzo da 2040 a 2047 sono situati i puntatori agli sprite; i valori contenuti in ciascuno di essi moltiplicati per 64 danno l'indirizzo dello sprite corrispondente. Il primo puntatore corrisponde allo sprite 1 e l'ultimo allo sprite 8. Abbiamo modificato il programma ES35, ottenendo ES35BIS, in modo che, invece di alternare nello sprite 1 le altre figure prelevandole dalle stringhe corrispondenti, modifichiamo il numero contenuto nel puntatore dello sprite 1, facendolo ciclicamente puntare alle altre figure che stanno nei buffer degli sprite.
Si ricordi che quando si interrompe il programma con RUN/STOPRESTORE i puntatori non sono nella situazione iniziale, per cui è megli eseguire in immediato questa linea:

J=56:FOR K=2040 TO 2047:POKE K,J:J=J+1:NEXT K

che rimette a posto le cose.

```
1 REM ES35BIS
3 REM CORRIDORE
5 REM COLORE SFONDO E BORDO
7 COLOR 0,1:COLOR 4,1
9 REM CICLO ATTESA SCAMBIO FIGURA E VELOCITA'
11 A=50:V=3
13 REM CARICA FILE FSPRC CON 8 SPRITE
15 BLOAD "FSPRC",B0
17 REM DISATTIVA TUTTI GLI SPRITE
19 FOR J=1 TO 7:SPRITE J,0: NEXT J
21 REM PULISCE IL VIDEO
```

```
23 PRINT"J"
25 REM ATTIVA PRIMO SPRITE
27 SPRITE 1,1,7,0,1,1,1
29 REM POSIZIONA PRIMO SPRITE
31 MOVSPR 1,32,150
33 REM AVVIA MOVIMENTO CORRIDORE
35 MOVSPR 1,90#V
37 REM CICLO ALTERNANZA FIGURE
39 FOR J=56 TO 62
41 POKE 2040,J:REM MODIFICA PUNTATORE A SPRITE
43 FOR I=1 TO A:NEXT I:REM CICLO ATTESA
45 NEXT J
47 GOTO 39
```

Nel paragrafo 3.2, dedicato alle mappe di memoria, si trovano elencati i registri del chip 8564, molti dei quali sono coinvolti nella gestione degli sprite. Per poter gestire direttamente questi registri bisogna tener presente che i primi 21 vengono ricaricati dalla routine IRQ prelevandone il contenuto dalla RAM del banco 0, e precisamente dall'indirizzo 4566 all'indirizzo 4587 ($11D6-$11EA). In conseguenza se si vuole lavorare direttamente sui registri del chip 8564, per i primi 21 è necessario utilizzare questi indirizzi, mentre per gli altri si usano gli indirizzi del blocco di I/O. Abbiamo preparato il programma ES36 per esemplificare questo tipo di gestione degli sprite. Il programma esegue i seguenti passi:
– carica il file FSPRB, inizializza i puntatori agli sprite, predispone i colori e azzera le variabili;
– pone nei registri del chip 8564 le coordinate X e Y iniziali deglli sprite 1 e 2, che devono mostrare due omini espansi che si guardano, situati alle due estremità laterali del video (coordinate 12,130 e 308,130);
– scrive nei registri del colore i colori degli sprite 1 e 2;
– abilita l'espansione verticale e orizzontale degli sprite 1 e 2;
– attiva gli sprite rendendoli visibili;
– in base al valore del flag SW, che inizia con 0, vengono posti alternativamente nei puntatori 1 e 2 i valori necessari per vedere gli omini che camminano e modificate le coordinate X degli omini in modo che si avvicinino tra loro;
– dopo ogni spostamento degli omini viene controllato il registro delle collisioni, e se si è verificata una collisione gli omini svengono, cioè si fanno puntare i puntatori 2040 e 2041 alle ultime due figure memorizzare nei buffer degli sprite;
– premendo un tasto vengono disabilitati gli sprite e ripristinati i valori di default nei puntatori.

```
1 REM ES36
3 REM OMINI CHE SI SCONTRANO
5 REM CARICA FILE FSPRB CON SPRITE
7 BLOAD"FSPRB",B0
9 REM SISTEMA PUNTATORI
11 GOSUB 95
13 REM SISTEMA COLORI
15 COLOR 0,1:COLOR 4,5:COLOR 5,2:COLOR 1,8
17 PRINT"J":CLR:REM AZZERA VARIABILI
```

```
19 REM COORDINATE INIZIALI NEI REGISTRI
21 REM DEGLI SPRITE 1 E 2
23 BANK 15:POKE 4566,12:POKE 4567,130
25 BANK 15:POKE 4568,52:POKE 4569,130
27 BANK 15:POKE 4582,2
29 REM COLORE SPRITE
31 BANK15:POKE 53287,PEEK(53287) AND 240 OR 2
33 BANK 15:POKE 53288,PEEK(53288) AND 240 OR 5
35 REM ABILITA ESPANSIONE ORIZZONTALE
37 BANK 15:POKE 53277,3
39 REM ABILITA ESPANSIONE VERTICALE
41 BANK 15:POKE 53271,3
43 REM SISTEMA PUNTATORI
45 POKE 2040,56:POKE 2041,60
47 REM ATTIVA SPRITE 1 E 2
49 BANK 15:POKE 53269,3
51 REM SPOSTAMENTO OMINI
53 REM MODIFICA PUNTATORI
55 IF SW=0 THEN POKE 2040,57:POKE 2041,61:SW=1:GOTO 59
57 POKE 2040,56:POKE 2041,60:SW=0
59 GOSUB 79
61 REM CONTROLLA COLLISIONE TRA SPRITE
63 IF PEEK(53278)=0 THEN 55
65 REM OMINI SVENUTI
67 POKE 2040,62:POKE 2041,63
69 GETKEY A$
71 REM RIPRISTINA PUNTATORI SPRITE
73 POKE 53269,0:REM DISABILITA SPRITE
75 GOSUB 95
77 STOP
79 REM MODIFICA COORDINATE X
81 A=PEEK(4582):X1=A AND 1:X2=A AND 2
83 X1=X1*256+PEEK(4566):X1=X1+4
85 X2=X2*128+PEEK(4568):X2=X2-4
87 POKE 4568,(X2 AND 255)
89 POKE 4566,(X1 AND 255)
91 POKE 4582,INT(X2/256)*2 OR INT(X1/256)
93 RETURN
95 REM INIZIALIZZAZIONE PUNTATORI SPRITE
97 J=56:FOR K=2040 TO 2047:POKE K,J:J=J+1:NEXT K
99 RETURN
```

Si noti l'utilizzo di due routine, una alla linea 95 per sistemare i puntatori agli sprite, e l'altra alla linea 79 per modificare le coordinate X degli sprite. Infatti, mentre le coordinate Y si mantengono in valori minori di 256, le coordinate X possono superare 255 e, in conseguenza il loro primo bit di sinistra, che vale 256, è memorizzato nel registro 53264 (registro ombra 4582) in posizione opportuna. Per calcolare le coordinate X è necessario ricavare il vecchio valore, modificarlo e separare la parte

HIGH dalla parte LOW; la routine delle linee 79-93 esegue questi calcoli, incrementando la coordinata dell'omino rosso di 4 e decrementando quella dell'omino verde di 4.
Si suggerisce di completare il programma ES36 aggiungendo una parte che, dopo un adatto intervallo di tempo, faccia rinvenire gli omini svenuti e li faccia arrampicare verso l'alto, utilizzando le due figure del file FSPRB che simulano questo movimento e che non abbiamo ancora utilizzato.

Come si puó vedere utilizzando i registri del chip 8564 è possibile individuare molto bene le collisioni tra gli sprite, ma indubbiamente un buon sfruttamento di queste possibilità si ottiene solo programmando in linguaggio macchina.

Il movimento e l'animazione delle figure sul video possono essere ottenuti anche in altro modo, con la tecnica di posizionare una figura, cancellarla e posizionarla abbastamnza vicino e ripetendo il ciclo con opportuni intervalli di tempo. Se si alternano le figure si ha l'animazione e non solo il movimento.

2.10 LA STAMPANTE

L'istruzione OPEN per la stampante ha al massimo tre parametri:

OPEN LFN, DN, SA

dove:
- LFN, se >127 provoca una doppia spaziatura tra le linee;
- DN, puó valere da 4 a 7 e dipende dalla stampante;
- SA, puó avere un particolare significato, che dipende dalla stampante. Per la stampante MPS 801: SA=0 produce il set di caratteri maiuscolo/grafico, SA=7 produce il set di caratteri minuscolo/maiuscolo.

Le istruzioni PRINT# LFN inviano alla stampante dati come variabili della lista di stampa, e codici di controllo come stringhe, che vengono interpretati dalla stampante per produrre effetti particolari come ingrandimento dei caratteri, passaggio al modo grafico, cambio del set di caratteri, caratteri in campo inverso, ripetizione di un carattere grafico, spostamento della posizione di stampa. Quello che si puó ottenere in stampa dipende dal tipo di stampante in uso. Si raccomanda di leggere con attenzione il manuale della stampante disponibile per poterla utilizzare al meglio.

Spesso si dirotta l'uscita del video alla stampante, eseguendo le seguenti istruzioni:

OPEN LFN,DN:CMD LFN

dopo di che le istruzioni PRINT senza il carattere'#' o LIST mandano i dati alla stampante.
Per interrompere il flusso di dati verso la stampante è necessario scrivere:

PRINT# LFN:CLOSE LFN

che chiude la linea di comunicazione e chiude il file.

Per poter eseguire la copia su carta del video per punti è necessario che la stampante accetti la stampa per punti (per esempio le stampanti a margherita non lo consentono); in tale caso si puó scandire il video e inviare le colonne di punti alla stampante. Nel libro dedicato al BASIC del COMMODORE 64 abbiamo riportato diversi programmi di questo tipo, che possono essere adattati per il COMMODORE 128.
Si deve tener presente che il video a 80 colonne ha la larghezza di 640 punti, mentre una stampante come la MPS 801 puó stampare su una linea solo 480 punti (corrispondenti a 80 caratteri larghi 6 punti); in conseguenza per poter eseguire la copia per punti del video a 80 colonne occorre una stampante che possa stampare più di 80 caratteri per linea.

2.11 I FILE SU CASSETTA E SU DISCO

La gestione dei file su cassetta dipende completamente dal calcolatore, mentre quella dei file su disco dipende anche dal software residente nella memoria ROM dell'unità a disco.

Le informazioni contenute sul manuale del COMMODORE 128 sono sufficienti per poter usare il registratore a cassetta sia per memorizzare programmi che file sequenziali di dati. In realtà utilizzare la cassetta come unica memoria di massa esterna per il COMMODORE 128 risulta un po' sproporzionato alla potenza del calcolatore.

Per quanto riguarda la gestione dei file su disco, per approfondire l'argomento è necessario leggere con attenzione il manuale venduto insieme all'unità a disco. Per parte nostra ci proponiamo di preparare un libro dedicato alle nuove unità 1570 e 1571. Se il lettore utilizza l'unità 1541 puó trovare utile il libro COMMODORE 64 I FILE, che la tratta diffusamente.
Il BASIC 7.0 mette a disposizione altre istruzioni per trattare i file su disco, oltre quelle già presenti nel BASIC 2.0; si posso utilizzare sia le une che le altre, come si preferisce.

CAPITOLO 3

UTILIZZO DELLA MEMORIA

3.1 GESTIONE DELLA MEMORIA

Il COMMODORE 128 si basa sul microprocessore 8502 che dispone di un BUS di indirizzamento a 16 linee e quindi puó indirizzare 64K di memoria, con indirizzi esadecimali da 0000 a FFFF (da 0 a 65535 in decimale). La prima cosa da chiedersi è come mai si chiama COMMODORE 128. La risposta è che la sua architettura comprende una unità per la gestione della memoria, chiamata MMU (Memory Management Unit), che fa lavorare il microprocessore con più di 64K di memoria, anche se esso puó indirizzare solo 64K per volta. Gli utilizzatori del COMMODORE 64 hanno già esperienza dell'utilizzo alternativo di parti di memoria; qui il sistema è molto più perfezionato.

In realtà il COMMODORE 128 dispone di 372K di memoria, come risulta dall'esame della figura 3.1, nella quale sono evidenziate tutte le parti di memoria che possono essere attive, ma mai tutte insieme. Nella figura 3.1 non sono presenti i 16K di memoria RAM del chip 8563, che sono gestibili in un modo un po' particolare. Per comprendere il significato della figura 3.1 dobbiamo vedere cosa fa l'unità MMU. Nel seguito, quando non chiarito dal contesto, i numeri esadecimali sono preceduti dal carattere $.

Nel COMMODORE 128 sono possibili 16 diverse configurazioni di memoria, in ognuna delle quali sono utilizzati in alternativa alcuni dei blocchi di memoria schematizzati nella figura 3.1.
Segue l'elenco delle configurazioni possibili, che possiamo chiamare banchi da un punto di vista logico, dato che sono formati da diversi blocchi, e che numeriamo in esadecimale da 0 a F. Il numero del banco in decimale, da 0 a 15, è quello che deve essere utilizzato nell'istruzione BANK del BASIC per selezionare una delle 16 configurazioni possibili.

BANCO 0 (decimale 0)

ind. esadecimali	ind. decimali	contenuto
0000 - FFFF	0 - 65535	RAM dal blocco 0 (meno parte MMU)

BANCO 1 (decimale 1)

ind. esadecimali	ind. decimali	contenuto
0000 - 03FF	0 - 1023	RAM dal blocco 0
0400 - FFFF	1024 - 65535	RAM dal blocco 1 (meno parte MMU)

BANCO 2 (decimale 2)

ind. esadecimali	ind. decimali	contenuto
0000 - 03FF	0 - 1023	RAM dal blocco 0
0400 - FFFF	1024 - 65535	RAM dal blocco 2 (meno parte MMU)

BANCO 3 (decimale 3)

ind. esadecimali	ind. decimali	contenuto
0000 - 03FF	0 - 1023	RAM dal blocco 0
0400 - FFFF	1024 - 65535	RAM dal blocco 3 (meno parte MMU)

BANCO 4 (decimale 4)

ind. esadecimali	ind. decimali	contenuto
0000 - 7FFF	0 - 32767	RAM dal blocco 0
8000 - CFFF	32768 - 53247	ROM funzioni interne
D000 - DFFF	53248 - 57343	I/O
E000 - FFFF	57344 - 65535	ROM funzioni interne (meno parte MMU)

BANCO 5 (decimale 5)

ind. esadecimali	ind. decimali	contenuto
0000 - 03FF	0 - 1023	RAM dal blocco 0
0400 - 7FFF	1024 - 32767	RAM dal blocco 1
0800 - CFFF	32768 - 53247	ROM funzioni interne
D000 - DFFF	53248 - 57343	I/O
E000 - FFFF	57344 - 65535	ROM funzioni interne (meno parte MMU)

BANCO 6 (decimale 6)

ind. esadecimali	ind. decimali	contenuto
0000 - 03FF	0 - 1023	RAM dal blocco 0
0400 - 7FFF	1024 - 32767	RAM dal blocco 2
0800 - CFFF	32768 - 53247	ROM funzioni interne
D000 - DFFF	53248 - 57343	I/O
E000 - FFFF	57344 - 65535	ROM funzioni interne (meno parte MMU)

BANCO 7 (decimale 7)

ind. esadecimali	ind. decimali	contenuto
0000 - 03FF	0 - 1023	RAM dal blocco 0
0400 - 7FFF	1024 - 32767	RAM dal blocco 3
0800 - CFFF	32768 - 53247	ROM funzioni interne
D000 - DFFF	53248 - 57343	I/O
E000 - FFFF	57344 - 65535	ROM funzioni interne (meno parte MMU)

BANCO 8 (decimale 8)

ind. esadecimali	ind. decimali	contenuto
0000 - 7FFF	0 - 32767	RAM dal blocco 0
8000 - CFFF	32768 - 53247	ROM funzioni esterne
D000 - DFFF	53248 - 57343	I/O
E000 - FFFF	57344 - 65535	ROM funzioni esterne (meno parte MMU)

BANCO 9 (decimale 9)

ind. esadecimali	ind. decimali	contenuto
0000 - 03FF	0 - 1023	RAM dal blocco 0
0400 - 7FFF	1024 - 32767	RAM dal blocco 1
0800 - CFFF	32768 - 53247	ROM funzioni esterne

D000 - DFFF	53248 - 57343	I/O
E000 - FFFF	57344 - 65535	ROM funzioni esterne (meno parte MMU)

BANCO A (decimale 10)

ind. esadecimali	ind. decimali	contenuto
0000 - 03FF	0 - 1023	RAM dal blocco 0
0400 - 7FFF	1024 - 32767	RAM dal blocco 2
0800 - CFFF	32768 - 53247	ROM funzioni esterne
D000 - DFFF	53248 - 57343	I/O
E000 - FFFF	57344 - 65535	ROM funzioni esterne (meno parte MMU)

BANCO B (decimale 11)

ind. esadecimali	ind. decimali	contenuto
0000 - 03FF	0 - 1023	RAM dal blocco 0
0400 - 7FFF	1024 - 32767	RAM dal blocco 3
0800 - CFFF	32768 - 53247	ROM funzioni esterne
D000 - DFFF	53248 - 57343	I/O
E000 - FFFF	57344 - 65535	ROM funzioni esterne (meno parte MMU)

BANCO C (decimale 12)

ind. esadecimali	ind. decimali	contenuto
0000 - 7FFF	0 - 32767	RAM dal blocco 0
8000 - BFFF	32768 - 49151	ROM funzioni interne
C000 - CFFF	49152 - 53247	ROM KERNAL
D000 - DFFF	53248 - 57343	I/O
E000 - FFFF	57344 - 65535	ROM KERNAL (meno parte MMU)

BANCO D (decimale 13)

ind. esadecimali	ind. decimali	contenuto
0000 - 7FFF	0 - 32767	RAM dal blocco 0
8000 - BFFF	32768 - 49151	ROM funzioni esterne
C000 - CFFF	49152 - 53247	ROM KERNAL
D000 - DFFF	53248 - 57343	I/O
E000 - FFFF	57344 - 65535	ROM KERNAL (meno parte MMU)

BANCO E (decimale 14)

ind. esadecimali	ind. decimali	contenuto
0000 - 3FFF	0 - 16383	RAM dal blocco 0
4000 - BFFF	16384 - 49151	ROM BASIC 7.0
C000 - CFFF	49152 - 53247	ROM KERNAL
D000 - DFFF	53248 - 57343	ROM caratteri
E000 - FFFF	57344 - 65535	ROM KERNAL (meno parte MMU)

BANCO F (decimale 15)

ind. esadecimali	ind. decimali	contenuto
0000 - 3FFF	0 - 16383	RAM dal blocco 0
4000 - BFFF	16384 - 49151	ROM BASIC 7.0
C000 - CFFF	49152 - 53247	ROM KERNAL
D000 - DFFF	53248 - 57343	I/O
E000 - FFFF	57344 - 65535	ROM KERNAL (meno parte MMU)

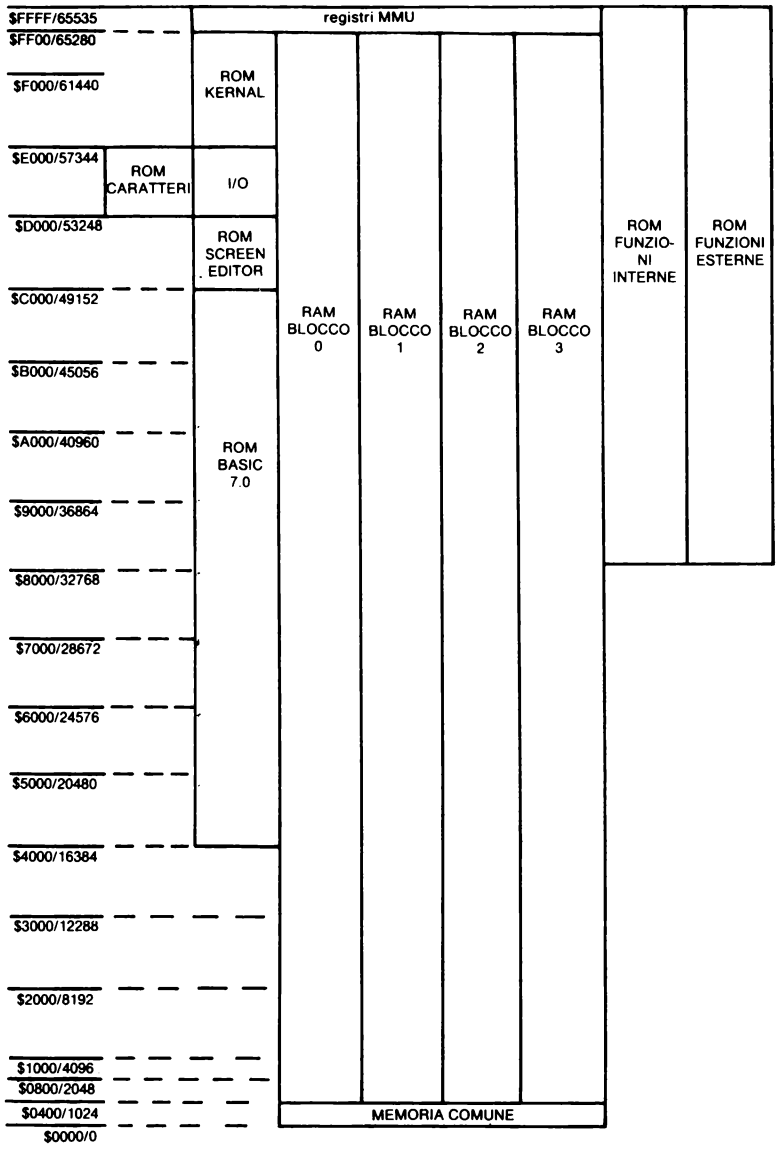


Fig. 3.1 Schema della memoria

Quando si accende il calcolatore senza alcuna predisposizione è attivo il banco logico 15 e possiamo programmare in BASIC 7.0 nel modo 128. Riportiamo nella figura 3.2 lo schema della memoria in questo modo di funzionamento.

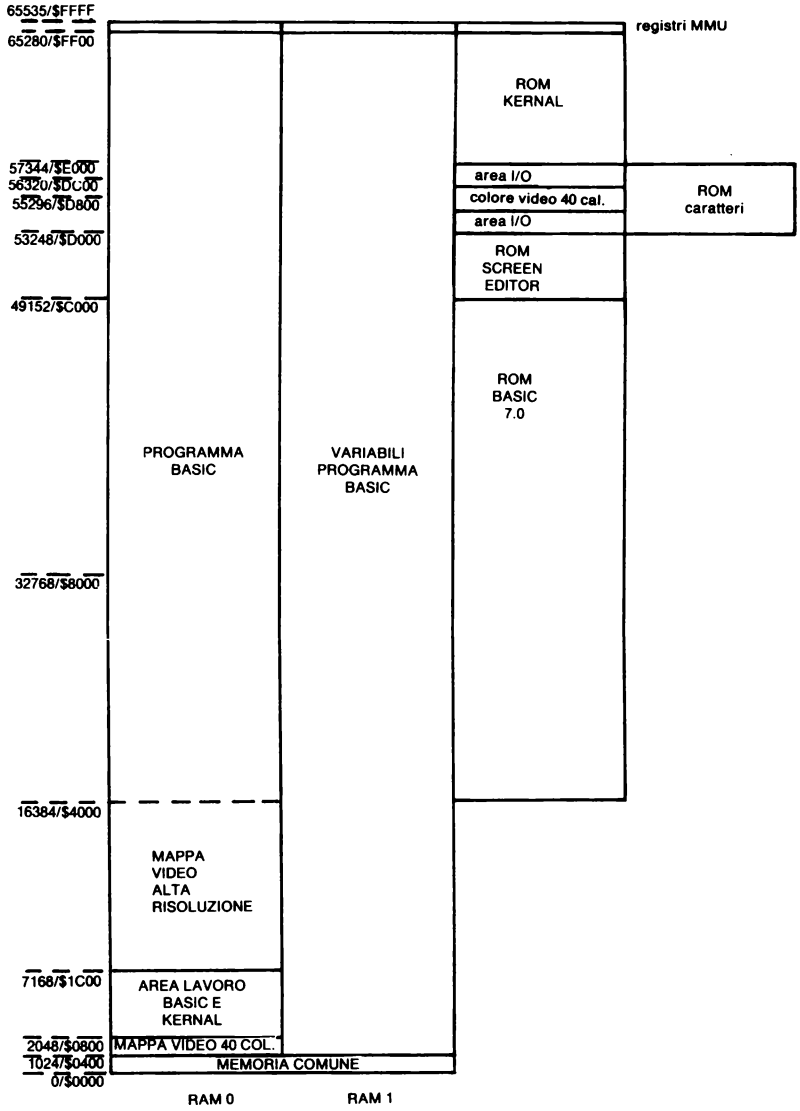


Fig. 3.2 Utilizzo della memoria nel modo 128 BASIC

Come si vede dalla figura 3.2 il testo del programma in BASIC sta nella memoria RAM del blocco 0, mentre le variabili stanno nella memoria RAM del blocco 1. Inoltre la memoria ROM dei caratteri corrisponde a indirizzi utilizzati anche per altri scopi; in conseguenza durante la preparazione e la prova di un programma in BASIC vengono alternativamente utilizzati i banchi logici 15, 14, 1 e 0, ma in modo trasparente per l'utente. Questo significa che provvede l'unità MMU a gestire l'utilizzo del banco che serve. Se invece l'utente vuole accedere da programma a un banco che puó essere diverso da quello attivo al momento, deve utilizzare l'istruzione BANK e poi tornare al banco 15, che è quello attivo per default. Quando in un programma BASIC si eseguono le istruzioni che agiscono direttamente sulla memoria, come POKE, PEEK o SYS, esse lavorano sugli indirizzi attivi nel banco 15, a meno di non rendere attivo un altro banco logico.
Dalla figura 3.2 si ricavano le seguenti informazioni:
– il programma BASIC, se non è attivo lo schermo ad alta risoluzione, inizia al byte 1C00 esadecimale (7168 in decimale);
– il programma BASIC, se è attivo lo schermo ad alta risoluzione, inizia al byte 4000 esadecimale (16384 in decimale);
– la mappa del video a 40 colonne inizia al byte 0800 esadecimale (2048 decimale);
– la mappa dei colori del video a 40 colonne inizia al byte D800 esadecimale (55296 decimale);
– i 256 byte di indirizzo più alto, da FF00 a FFFF esadecimale (65280-65535 decimale), e i primi 1024 byte (quelli di indirizzo più basso, da 0000 a 0399 esadecimale (0-1023 decimale) sono utilizzati allo stesso modo nei diversi banchi, cioè sono sempre presenti qualunque sia la configurazione di banco logico attivo.

Si tenga presente che la mappa dei colori utilizza solo il semibyte di destra di ogni locazione (nibble), in conseguenza per leggere correttamente un colore si deve scrivere:

```
BANK 15:N = PEEK(K) AND 15
```

dove K varia da 55296 a 56319, e BANK 15 dà la sicurezza di leggere dal banco logico 15, dove è attivo il blocco di I/O.
Analogamente per scrivere correttamente un colore si utilizza la sequenza:

```
BANK 15: POKE K, (PEEK(K) AND 240 OR N)
```

dove N è il numero del colore desiderato per i caratteri.
Per modificare il colore del bordo si puó agire direttamente nel blocco di I/O con la sequenza:

```
BANK 15:POKE 53280, (PEEK(53280) AND 240 OR N)
```

e per modificare il colore dello sfondo:

```
BANK 15:POKE 53281, (PEEK(53281) AND 240 OR N)
```

dove N è il numero del colore desiderato.

Dopo un RESET del calcolatore o all'accensione, in modo 128, possiamo leggere con la funzione PEEK il valore di alcuni puntatori situati nella pagina zero (PRINT PEEK(n) + 256*PEEK(n+1)); troviamo:
45-46, puntatore all'inizio del programma BASIC, contiene 7169 (con video a 40

colonne in modo testo);
47-48, puntatore a inizio variabili nel banco 1, contiene 1024;
49-50, puntatore a inizio array nel banco 1, contiene 1024;
51-52, puntatore a inizio memoria libera nel banco 1, contiene 1024 (i tre precedenti valori sono uguali dato che non è ancora presente alcuna variabile);
53-54, puntatore a inizio zona dinamica per le stringhe nel banco 1, contiene 65280 (si ricordi che tale zona è gestita per indirizzi decrescenti);
57-58, puntatore alla fine della zona dinamica per le stringhe nel banco 1, contiene 65280, infatti non sono presenti stringhe.
I valori dei puntatori alle diverse zone delle variabili nel banco 1 si modificano quando viene scritto e utilizzato un programma BASIC; in tutto nel banco 1 sono disponibili per le variabili 65280-1024=64256 byte.
Nel banco 0, se il video è in modo testo, il programma BASIC inizia nel byte 7168 e puó arrivare al byte 65279, cioè puó occupare 65280-7168=58112 byte. Se invece il video è in modo grafico il programma BASIC inizia nel byte 16384. Il numero dei byte liberi (122365) che compare nel messaggio iniziale risulta dal seguente calcolo:

spazio variabili banco 1 + spazio programma banco 0 - 3 byte (1 di inizio programma e 2 di fine programma posti a zero dal sistema), cioè:

64256 + 58112 - 3 = 122365.

Esaminando la configurazione dei banchi logici si vede che il banco 2 è identico al banco 0 e che il banco 3 è identico al banco 1, salvo che fanno riferimento rispettivamente ai blocchi di RAM 2 e 3, invece che 0 e 1. In effetti i blocchi di memoria RAM 2 e 3 non sono installati e risultano identici ai blocchi 0 e 1 fino a quando non saranno disponibili altre espansioni.
I banchi logici da 4 a 7 assumono interesse se l'utente aggiunge internamente memorie ROM personalizzate che vanno ad occupare gli spazi denominati 'ROM funzioni interne'.
I banchi logici da 8 a 11 contengono lo spazio per le memorie ROM esterne, cioè assumono interesse se si utilizzano cartridge inserendoli nelle porte di espansione.
I banchi 12 e 13 utilizzano 32K di memoria RAM dal blocco 0 e 16K di memoria ROM, rispettivamente interna o esterna, oltre alle routine KERNAL e al blocco di I/O; essi hanno interesse se si utilizza una memoria ROM interna o un cartridge.

Nel presentare l'elenco delle configurazioni possibili abbiamo riportato sempre tra parentesi la frase 'meno parte MMU' e, inoltre, nelle due figure 3.1 e 3.2 compaiono nella parte bassa e nella parte alta della memoria due zone comuni. In effetti non sarebbe possibile operare i frequenti scambi di zone di memoria attive senza avere alcune parti di memoria che rimangono sempre a disposizione.
Nel COMMODORE 128 sono presenti due chip, lo MMU (Memory Management Unit) e il PLA (Programmable Logic Array), che provvedono alla gestione degli indirizzi della memoria consentendo le configurazioni dei sedici banchi logici già elencati. Si deve inoltre tener presente che quanto abbiamo visto riguarda il modo 128, ma che il sistema puó funzionare anche nel modo 64 e nel modo CP/M, e che in quest'ultimo caso viene attivato anche il microprocessore Z80.

Nella parte alta della memoria sono presenti i registri principali per la gestione MMU; essi hanno gli indirizzi esadecimali da FF00 a FF04 (65280 - 65284 decimale) e sono adibiti allo stesso scopo in tutti i blocchi. Oltre a questo set di registri ne è disponibile un altro agli indirizzi esadecimali del blocco di I/O, D500 - D50B

(54528 - 54539 decimale), che ha i primi 5 registri con lo stesso significato dei precedenti e gli ultimi sei con altri significati.

Il primo registro, di indirizzo esadecimale FF00 e D500 (65280 e 54528 decimale), definisce il banco logico attivo. Il significato dei bit di questo registro è il seguente:

bit	contenuto	
7 e 6	blocco RAM:	00 per blocco 0 01 per blocco 1 10 per blocco 2 (ora duplicato blocco 0) 11 per blocco 3 (ora duplicato blocco 1)
5 e 4	indirizzi da	C000 a CFFF esadecimali (49152 - 53247 decimali)
	e da	E000 a EFFF esadecimali (57344 - 61439 decimali)
	00 per ROM	KERNAL
	01 per ROM	funzioni interne
	10 per ROM	funzioni esterne
	11 per RAM	
3 e 2	indirizzi da	8000 a BFFF esadecimali (32768 - 49151 decimali)
	00 per ROM	BASIC parte alta
	01 per ROM	funzioni interne
	10 per ROM	funzioni esterne
	11 per RAM	
1	indirizzi da	4000 a 7FFF esadecimali (16384 - 32767 decimali)
	0 per ROM	BASIC parte bassa
	1 per RAM	
0	indirizzi da	D000 a DFFF esadecimali (53248 - 57343 decimali)
	0 per I/O	
	1 per ROM	caratteri o RAM

Il contenuto di questo registro risulta in decimale:

0 (00 esadecimale) per il banco 15 (binario 00 00 00 0 0)
1 (01 esadecimale) per il banco 14 (binario 00 00 00 0 1)
127 (7F esadecimale) per il banco 1 (binario 01 11 11 1 1)
63 (3F esadecimale) per il banco 0 (binario 00 11 11 1 1).

La memorizzazione del valore desiderato puó essere operata o su FF00 o su D500 (FF00 risulta sempre disponibile, mentre il set dei registri D5ii (dove 'ii' indica le due cifre esadecimali finali del registro) è disponibile solo nel banco 15 dove è attivo il blocco di I/O) ed ha lo stesso effetto.
I registri da D501 a D504 vengono chiamati *registri di preconfigurazione* e sono gestiti in un modo particolare; in essi puó essere preparata una configurazione di banco logico che diventa attiva quando viene trasferita nel corrispondente registro

FFii nella parte alta della memoria.
Al momento dell'accensione o dopo un RESET, se chiamiamo il MONITOR e leggiamo il contenuto dei 4 registri FFii con il comando M FF01 FF04, troviamo: 3F 7F 01 41 cioe' i primi tre registri contengono il valore da trasferire in FF00 per rendere attivi rispettivamente i banchi 0, 1 e 14. L'operazione di lettura dei registri FFii fornisce i valori precedentemente contenuti nei registri D5ii. Per contro una operazione di memorizzazione di un qualunque numero in uno dei registri con indirizzo da FF01 a FF04 ha per effetto di trasferire in esso il valore contenuto nel corrispondente registro D5ii, qualunque sia il numero utilizzato nell'istruzione di memorizzazione, che resta inutilizzato. Per rendere attivo un banco logico si puó agire sul registro FF00, oppure su uno dei registri FFii, che si chiamano rispettivamente LCRA, LCRB, LCRC e LCRD, dopo essersi assicurati che il corrispondente registro di preconfigurazione contenga il giusto valore, oppure eseguire l'istruzione BASIC BANK N, con N che puó valere da 0 a 15. Il registro FF00 si chiama CR e lo stesso nome viene attribuito anche al registro D500. I registri da D501 a D504 si chiamano rispettivamente PCRA, PCRB, PCRC e PCRD.

Il registro di indirizzo esadecimale D505 (decimale 54533) si chiama MCR e definisce il modo nel quale lavora il calcolatore (all'inizio in modo 128 esso contiene 183). I suoi bit hanno il seguente significato:
- Bit 0: 0 significa processore Z80 attivo;
  1 significa processore 8502 attivo.
- Bit 1: non usato.
- Bit 2: non usato.
- Bit 3: controlla la trasmissione seriale veloce.
- Bit 4, 5 e 6: indicano la presenza di memorie ROM esterne in modo 64.
- Bit 7: 1 indica schermo a 40 colonne;
  0 indica schermo a 80 colonne.

Il registro di indirizzo esadecimale D506 (decimale 54534) si chiama RCR e definisce la configurazione della memoria RAM in uso (all'inizio esso contiene 4). Il significato dei suoi bit è il seguente:
- Bit 0 e 1: definiscono quanta memoria RAM è usata in comune dai banchi logici. Il valore 00 corrisponde a 1K di memoria RAM comune, il valore 01 a 4K, il valore 10 a 8 K e il valore 11 a 16K.
- Bit 2 e 3: definiscono dove è localizzata la memoria comune, se nella parte bassa e/o nella parte alta. Nel modo 128 lo stato di questi bit denota la presenza di memoria comune agli indirizzi bassi.
- Bit 4 e 5: non usati.
- Bit 6 e 7: definiscono in quale blocco è localizzata la memoria RAM utilizzata dal video chip. La mappa del video puó trovarsi nel blocco 0 o nel blocco 1, ma la mappa del colore rimane sempre nel blocco I/O agli indirizzi esadecimali D800 - DBFF (55296 - 56319 decimali).

I quattro registri di indirizzo esadecimale D507, D508, D509 e D50A sono i puntatori alle pagine di memoria RAM 0 e 1, e consentono di rilocare tali pagine sia in modo 128 che in modo CP/M.
Il registro di indirizzo esadecimale D507 (54535 decimale), di nome P0L, è il puntatore alla pagina 0, parte LOW, e in esso devono essere memorizzate le prime due cifre esadecimali del nuovo indirizzo che si vuole assegnare alla pagina zero. Le ultime due cifre esadecimali vanno da 00 a FF. Il registro di indirizzo D508, di nome P0H, parte HIGH, non viene utilizzato nell'attuale configurazione del calcolatore. Rilocare la pagina zero significa che ogni riferimento ad indirizzi

assoluti da 0000 a 00FF agisce sulle corrispondenti posizioni della pagina rilocata. Si ricordi peró che i riferimenti ai nuovi indirizzi assoluti vanno evitati perché a loro volta agiscono sugli indirizzi della pagina zero (vecchia); si ha cioè uno scambio di indirizzi. Gli unici indirizzi della pagina zero che non possono essere rilocati sono i primi due, 0000 e 0001.
Gli altri due registri, di indirizzo esadecimale D509 e D50A, rispettivamente di nome P1L e P1H, agiscono in modo analogo per la rilocazione della pagina 1 di memoria (di indirizzi 0100 - 01FF), che è molto importante dato che in essa si trova l'area stack.

L'ultimo registro del set considerato, di indirizzo esadecimale D50B (54539 decimale), contiene indicazioni sulla versione dello MMU e sulla quantità di memoria RAM presente. Nel COMMODORE 128 ora disponibile tale registro contiene il numero esadecimale 20 (32 decimale), a indicare la versione 0 di MMU e la presenza di 2 blocchi di 64K di memoria RAM.

All'accensione il calcolatore svolge i seguenti passi:

– 1. Controlla se le linee GAME o EXROM delle porte di espansione sono collegate, cioè se è presente un cartridge del 64, e, in caso affermativo, entra in modo 64.
– 2. Controlla se è presente un cartridge 128 agli indirizzi $8000 e/o $C000 sia per funzioni interne che per funzioni esterne. Se le memorie ROM hanno incorporata la partenza automatica esse cominciano a lavorare, altrimenti esse vengono prese in considerazione più avanti nell'ordine $8000 esterno, $C000 esterno, $8000 interno e $C000 interno (si vedano le configurazioni dei banchi logici 4/7 12 8/11 e 13).
– 3. Controlla se il tasto SHIFT è premuto, e, se si, si porta in modo 128, ma sotto controllo del MONITOR.
– 4. Controlla se il tasto COMMODORE è premuto, e, se si, si porta in modo 64 BASIC.
– 5. Controlla se il disco presente ha nel settore 0 della prima traccia le indicazioni di 'autoboot', se no si porta in modo 128.
– 6. Se presente un floppy con 'autoboot' controlla se è CP/M, e, se si, dà il controllo al processore Z80, viene caricato da disco il CPM ed esce il messaggio del CP/M.
– 7. Se presente un floppy con 'autoboot', ma non è CP/M, resta in modo 128, carica da disco quanto contenuto e comincia a lavorare.

Una delle operazioni compiute durante la fase di inizializzazione è la copiatura nella memoria RAM alta, comune a tutti i banchi, della parte di memoria ROM necessaria per il funzionamento del sistema. Tale parte comprende i registi dello MMU già visti, le routine necessarie per gestire gli interrupt e il reset del sistema e i relativi vettori e le routine per il passaggio al CP/M.
In particolare, durante la fase di inizializzazione, vengono copiati nei tre byte di indirizzo $FFF5,$FFF6 e $FFF7 del banco 1 i codici ASCII della sigla CBM ($43, $42 e $4D); il sistema controlla questi tre byte in caso di RESET e se trova la sigla CBM esegue un reset più veloce mediante un salto indiretto tramite il vettore di indirizzo $FFF8. Il contenuto iniziale dei due byte $FFF8 e $FFF9 è $24 e $E2, cioè l'indirizzo $E224, che conduce al reset del sistema in modo 128. Se i tre byte non contengono i codici ASCII della sigla CBM il sistema opera un reset completo utilizzando la routine che inizia in $FF3D.
E' possibile modificare il contenuto del vettore di indirizzo $FFF8 in modo che il calcolatore in caso di reset veloce (cioè quando trova la sigla CBM) passi nel modo

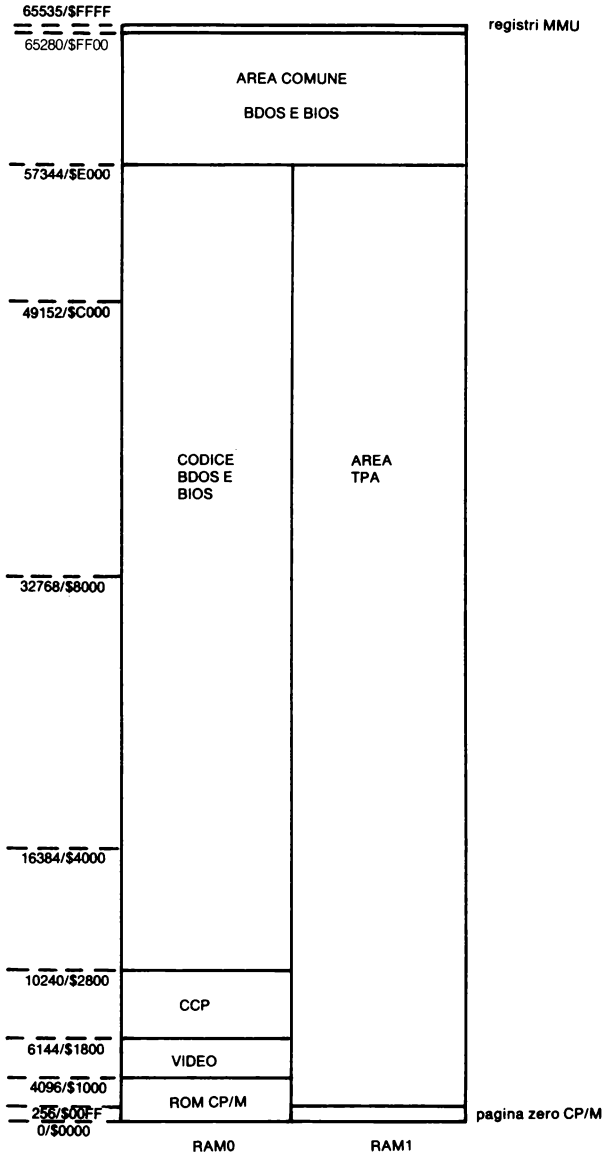


Fig. 3.3 Utilizzo dei banchi di memoria 0 e 1 in CP/M

64; basta entrare in MONITOR e scrivere:
> 1FFF8 4D FF
in modo che nel banco 1 il contenuto di $FFF8 diventa $4D e il contenuto di $FFF9 diventa $FF. In tale caso l'indirizzo $FF4D conduce al reset del sistema in modo 64. Si ricordi che per ottenere di nuovo il modo 128 in caso di reset si deve spegnere il calcolatore.

Il banco 15 differisce dal banco 14 per la presenza del blocco di I/O al posto della memoria ROM dei caratteri. Il COMMODORE 128 usa il sistema 'I/O mappato in memoria'; questo significa che ai registri necessari per la comunicazione delle periferiche con il calcolatore corrispondono indirizzi di memoria che possono essere trattati come tutti gli altri.

Quando il calcolatore funziona con video a 40 colonne è attivo il chip 8564, che dispone di 49 registri. Esso gestisce il video in modo testo a 40 colonne, gli sprite e la grafica ad alta risoluzione, inoltre è responsabile del funzionamento in modo FAST o SLOW e gestisce la penna ottica.

Quando invece il calcolatore funziona con video a 80 colonne è attivo il chip 8563, che dispone di 37 registri e di un suo blocco di memoria RAM di 16K; a questa memoria è possibile accedere tramite due registri del chip 8563. Nel modo a 80 colonne non sono disponibili le istruzioni grafiche del BASIC e per ottenere della grafica in modo veloce è necessario programmare in linguaggio macchina, ma si possono ottenere buoni risultati anche in BASIC. Il chip 8563 gestisce anche la penna ottica nel modo a 80 colonne.

Il chip SID dispone di 29 registri e gestisce il suono e i registri di controllo dei paddle.

I due chip CIA (6526 Complex Interface Adapter) svolgono un gran numero di funzioni; ognuno di essi dispone di 16 registri. Dal CIA 1 dipendono la lettura della tastiera, dei joystick, del mouse, dei paddle, la trasmissione seriale veloce con l'unità a disco 1571 e l'emissione di un segnale di interruzione IRQ ogni sessantesimo di secondo. Il CIA 2 controlla la trasmissione seriale a velocità normale, l'interfaccia RS-232, l'emissione di un segnale di interruzione NMI, la porta utente, la determinazione del blocco di memoria di 16K gestibile dal chip VIC 6584.

Nella figura 3.3 è mostrato lo schema di utilizzo dei blocchi di memoria 0 e 1 da parte del CP/M.

3.2 MAPPE DELLA MEMORIA

Riportiamo le mappe della memoria facendo riferimento alle zone indicate nelle figure 3.1 e 3.2. I numeri in notazione esadecimale, quando non chiarito dal contesto, sono preceduti dal carattere $.

ind. dec.	ind. esad.	contenuti
0-255	0000-00FF	PAGINA ZERO
0	0000	registro della direzione dei dati della porta di I/O del processore 8502

1	0001	registro dei dati della porta di I/O del processore 8502
2	0002	numero banco per routine JMPFAR e JSRFAR
3-4	0003-0004	contatore programma per JMPFAR e JSRFAR
5	0005	registro stack per JMPFAR e JSRFAR
6	0006	accumulatore per JMPFAR e JSRFAR
7	0007	registro X per JMPFAR e JSRFAR
8	0008	registro Y per JMPFAR e JSRFAR
9	0009	stack pointer per JMPFAR e JSRFAR
10	000A	carattere da ricercare
11	000B	indicatore per ricerca virgolette di chiusura stringhe
12	000C	posizione carattere dopo ultimo TAB
13	000D	puntatore buffer input, numero dimensioni variabili con indice
14	000E	dimensioni variabile con indice
15	000F	tipo dato: $FF=stringa, $00=numero
16	0010	tipo dato: $00=floating, $80=intero
17-18	0011-0012	indicatori a uso multiplo
19	0013	flag: $00=INPUT, $40=GET, $98=READ
20	0014	segno/confronto risultati
21	0015	numero canale
22-23	0016-0017	numero intero (conversione indirizzo da formato esponenziale)
24	0018	puntatore stack stringhe
25-26	0019-001A	indirizzo ultima stringa
27-35	001B-0023	stack per stringhe
36-39	0024-0027	puntatori vari
40-44	0028-002C	risultato moltiplicazione floating
45-46	002D-002E	puntatore all'inizio del programma in BASIC nel banco 0
47-48	002F-0030	puntatore all'inizio delle variabili nel banco 1
49-50	0031-0032	puntatore all'inizio degli ARRAY nel banco 1
51-52	0033-0034	puntatore all'inizio della memoria libera nel banco 1
53-54	0035-0036	puntatore all'inizio della zona dinamica per le stringhe nel banco 1
55-56	0037-0038	puntatore alla stringa usata per ultima nel banco 1 (valore del precedente -2)
57-58	0039-003A	puntatore alla fine della zona dinamica per le stringhe nel banco 1
59-60	003B-003C	numero linea attuale del programma BASIC
61-62	003D-003E	puntatore al carattere attuale del testo del programma BASIC
63-64	003F-0040	usato da PRINT USING
65-66	0041-0042	linea attuale per DATA
67-68	0043-0044	puntatore all'elemento attuale della linea DATA
69-70	0045-0046	puntatore per routine INPUT
71-72	0047-0048	puntatore al nome della variabile BASIC attuale

73-74	0049-004A	puntatore al dato della variabile attuale
75-76	004B-004C	puntatore alla variabile di controllo del ciclo FOR/NEXT
77-98	004D-0062	area di lavoro
99-104	0063-0068	FAC1, accumulatore 1 floating-point
105	0069	indicatore per valutazione segno
106-111	006A-006F	FAC2, accumulatore 2 floating-point
112	0070	segno risultato confronto FAC1 e FAC2
113	0071	ultimo byte FAC1 per arrotondamento
114-115	0072-0073	puntatore al buffer della cassetta
116-117	0074-0075	per AUTO, intervallo numeri linea
118	0076	flag per memoria riservata
119	0077	USING, per zeri riempimento memoria per sprite
120	0078	contatore
121	0079	memoria lavoro
122-124	007A-007C	situazione disco per DS$
125-126	007D-007E	limite run time stack
127	007F	flag diretto/indiretto
128-130	0080-0082	parola di stato per interprete
131	0083	colore attuale selezionato
132	0084	multicolor-1
133	0085	multicolor-2
134	0086	colore caratteri
135-136	0087-0088	fattore di scala per x
137-138	0089-008A	fattore di scala per y
139-143	008B-008F	utilizzata da PAINT
144	0090	byte di stato operazioni I/O
145	0091	flag STOP (127 per RUN/STOP premuto)
146	0092	controllo velocità nastro
147	0093	indicatore: 0=LOAD, 1=VERIFY
148	0094	flag per carattere nel buffer seriale
149	0095	buffer seriale carattere
150	0096	sincronizzazione cassetta
151	0097	area lavoro
152	0098	numero file aperti (puntatore tabella)
153	0099	periferica principale di input
154	009A	periferica principale di output
155	009B	parità cassetta
156	009C	switch cassetta
157	009D	flag dei messaggi: 192=controllo ed errore KERNAL 128=solo controllo 64=solo errore 0=nessun messaggio
158	009E	indicatore errore passo 1 nastro
159	009F	indicatore errore passo 2 nastro
160-162	00A0-00A2	orologio aggiornato 60 volte al secondo
163-171	00A3-00AB	area lavoro per cassetta
172-173	00AC-00AD	puntatore per LOAD, SAVE, VERIFY
174-175	00AE-00AF	indirizzo finale per LOAD, SAVE e VERIFY
176-177	00B0-00B1	costanti per operazioni cassetta
178-179	00B2-00B3	puntatore al buffer della cassetta
180-182	00B4-00B6	area lavoro RS-232
183	00B7	lunghezza nome file in uso

184	00B8	indirizzo logico file in uso
185	00B9	indirizzo secondario file in uso
186	00BA	numero logico periferica in uso
187-188	00BB-00BC	indirizzo nome del file in uso
189-192	00BD-00C0	area lavoro cassetta
193-194	00C1-00C2	ind. iniziale per LOAD, SAVE e VERIFY
195-196	00C3-00C4	ind. iniziale memoria per LOAD e SAVE
197	00C5	dato letto o scritto su nastro
198	00C6	banco dell'operazione in corso di LOAD, SAVE e VERIFY
199	00C7	banco del nome file in uso
200-201	00C8-00C9	puntatore al buffer di input RS-232
202-203	00CA-00CB	puntatore al buffer di output RS-232
204-205	00CC-00CD	puntatore alla tabella di scansione della tastiera
206-207	00CE-00CF	puntatore stringa per routine PRIMM
208	00D0	numero di caratteri nel buffer tastiera
209-210	00D1-00D2	per tasti funzione
211	00D3	stato tasti SHIFT, CONTROL, COMMODORE e ALT
212	00D4	coordinate tasto premuto
213	00D5	coordinate tasto precedente premuto
214	00D6	flag CR
215	00D7	flag schermo: 0 per 40, 128 per 80 col.
216	00D8	flag modo testo/grafico: 224 per GRAPHIC 4, 160 per GRAPHIC 3, 96 per GRAPHIC 2, 32 per GRAPHIC 1, 0 per GRAPHIC 0 (TEXT)
217	00D9	utilizzato da routine CHREN: 4 per I/O ind. $D000-$DFFF, 0 per ROM caratteri ind. $D000-$DFFF
218-223	00DA-00DF	area lavoro routine EDITOR
224-225	00E0-00E1	puntatore alla linea attuale di testo sul video
226-227	00E2-00E3	puntatore all'attributo colore della linea attuale
228	00E4	limite inferiore finestra video
229	00E5	limite superiore finestra video
230	00E6	margine sinistro finestra video
231	00E7	margine destro finestra video
232	00E8	colonna attuale del video
233	00E9	inizio linea attuale del video
234	00EA	fine linea attuale del video
235	00EB	linea attuale del cursore
236	00EC	colonna attuale del cursore
237	00ED	numero massimo linee del video
238	00EE	numero massimo di colonne per linea
239	00EF	ultimo carattere visualizzato
240	00F0	precedente carattere visualizzato
241	00F1	attributo ultimo carattere visualizzato
242	00F2	attributi conservati
243	00F3	flag RVS ON/OFF (RVS ON se <> 0)
244	00F4	flag QUOTE (QUOTE ON se <> 0)
245	00F5	flag INSERT ON (INSERT ON <> 0)

246	00F6	flag AUTO-INSERT
247	00F7	flag abilitazione alternanza tra i due set di caratteri: 128 disabilita, 0 abilita
248	00F8	flag abilitazione scrolling del video: 128 disabilita, 0 abilita
249	00F9	flag abilitazione CTRL/G
250-255	00FA-00FF	non usati, disponibili per utente

ind. dec.	ind. esad.	contenuti
256-511	0100-01FF	PAGINA UNO: stack del sistema

ind. dec.	ind. esad.	contenuti
512-1023	0200-03FF	RAM COMUNE, VETTORI E ROUTINE
512-673	0200-02A1	buffer di input del BASIC, 161 byte
674-686	02A2-02AE	INDFET, routine per leggere un carattere da qualunque banco
687-701	02AF-02BD	INDSTA, routine per memorizzare un carattere in qualunque banco
702-716	02BE-02CC	INDCPM, routine per confrontare caratteri in qualunque banco
717-738	02CD-02E2	JSRFAR, routine per saltare a un sottoprogramma in qualunque banco e tornare al banco di partenza
739-761	02E3-02F9	JMPFAR, routine per saltare a una routine in qualunque banco senza ritorno
768-769	0300-0301	IERROR, vettore per i messaggi di errore del BASIC
770-771	0302-0303	IMAIN, vettore per ciclo BASIC in modo immediato
772-773	0304-0305	ICRNCH, vettore utilizzato dalla routine che trasforma in codice intermedio una linea BASIC (da caratteri a token)
774-775	0306-0307	IQPLOP, vettore utilizzato dalla routine che trasforma i token in caratteri
776-777	0308-0309	IGONE, vettore per la routine che esegue un token
778-779	030A-030B	IEVAL, vettore utilizzato dalla routine che valuta un'espressione aritmetica
780-781	030C-030D	vettore per la routine che trasforma due byte in un token
782-783	030E-030F	vettore per la routine che lista un token nato da due byte
784-785	0310-0311	vettore per la routine che esegue un token nato da due byte
786-787	0312-0313	
788-789	0314-0315	CINV, vettore interrupt IRQ
790-791	0316-0317	CBINV, vettore interrupt BRK
792-793	0318-0319	NMINV, vettore interrupt NMI
794-795	031A-031B	IOPEN, vettore routine Kernal OPEN
796-797	031C-031D	ICLOSE, vettore routine Kernal CLOSE
798-799	031E-031F	ICHKIN, vettore routine Kernal CHKIN
800-801	0320-0321	ICKOUT, vettore routine Kernal CKOUT

802-803	0322-0323	ICLRCH, vettore routine Kernal CHRCH
804-805	0324-0325	IBASIN, vettore routine Kernal BASIN
806-807	0326-0327	IBSOUT, vettore routine Kernal BSOUT
808-809	0328-0329	ISTOP, vettore routine Kernal STOP
810-811	032A-032B	IGETIN, vettore routine Kernal GETIN
812-813	032C-032D	ICLALL, vettore routine Kernal CLALL
814-815	032E-032F	IMONITOR, vettore routine Kernal MONITOR
816-817	0330-0331	ILOAD, vettore routine Kernal LOAD
818-819	0332-0333	ISAVE, vettore routine Kernal SAVE
820-821	0334-0335	vettore carattere con CTRL
822-823	0336-0337	vettore carattere con SHFT
824-825	0338-0339	vettore carattere con ESC
826-827	033A-033B	vettore tasto premuto
828-829	033C-033D	vettore memorizzazione carattere
830-831	033E-033F	vettore decodifica
832-841	0340-0349	
842-851	034A-0353	buffer input tastiera
852-865	0354-0361	
866-875	0362-036B	tabella numeri logici file aperti
876-885	036C-0375	tabella numeri logici periferiche
886-895	0376-037F	tabella indirizzi secondari file
896-926	0380-039E	CHRGET, routine per leggere il carattere successivo del testo BASIC nel banco 0
902	0386	CHRGOT, punto entrata in CHRGET per il carattere precedente
927-938	039F-03AA	ricerca indiretta routine in ROM del banco 0
939-950	03AB-03B6	ricerca indiretta routine in ROM del banco 1
951-959	03B7-03BF	ricerca routine in ROM dal banco 1, usa come puntatore $24 e $25
960-968	03C0-03C8	ricerca routine in ROM dal banco 0, usa come puntatore $26 e $27
969-977	03C9-03D1	ricerca routine in ROM dal banco 0, usa come puntatore $3D e $3F

ind. dec.	ind. esad.	contenuti
1024-2047	0400-07FF	MAPPA VIDEO PER TESTO 40 COLONNE

ind. dec.	ind. esad.	contenuti
2048-7167	0800-1BFF	BANCO 0 - AREA LAVORO KERNAL E BASIC
2048-2559	0800-09FF	area stack BASIC
2560-2561	0A00-0A01	vettore reset sistema, punta routine partenza a caldo del BASIC
2562	0A02	flag che indica che il vettore del sistema è stato inizializzato
2563	0A03	flag per video, 0=NTSC, 1=PAL
2564	0A04	usato da CINT e da IOINIT
2565-2566	0A05-0A06	limite inferiore memoria utilizzata nel banco 0 per testo programma BASIC
2567-2568	0A07-0A08	limite superiore memoria utilizzata nel banco 1 per le variabili BASIC
2576	0A10	registro controllo RS-232
2577	0A11	registro comandi RS-232

2578-2579	0A12-0A13	baud-rate per RS-232
2580	0A14	registro di stato RS-232
2582-2583	0A16-0A17	costante tempo velocità RS-232
2584	0A18	puntatore all'ultimo carattere nel buffer di input RS-232
2585	0A19	puntatore al primo carattere nel buffer di input RS-232
2586	0A1A	puntatore all'ultimo carattere nel buffer di output RS-232
2587	0A1B	puntatore al primo carattere nel buffer di output RS-232
2589-2591	0A1D-0A1F	TIMER, decrementato 60 volte al secondo
2592	0A20	numero massimo di caratteri nel buffer della tastiera
2594	0A22	abilitazione ripetizione tasti: 128 tutti, 64 nessuno, 0 spazio, INS/DEL, movimento cursore
.2604	0A2C	bit 7/4: numero da moltiplicare per 1024 per ottenere ind. mappa video bit 3/0: numero da moltiplicare per 1024 per ottenere ind. memoria caratt. nel blocco di 16K visto dal chip 8564
2624-2687	0A40-0A7F	variabili screen-editor durante cambio 40/80 colonne (da $00E0)
2752	0AC0	numero ROM di funzioni presenti
2753-2756	0AC1-0AC4	tavola identificazione ROM di funzioni
2816-3007	0B00-0BBF	buffer cassetta
3072-3327	0C00-0CFF	buffer input RS-232
3328-3583	0D00-0DFF	buffer output RS-232
3584-4095	0E00-0FFF	area definizione sprite
4096-4105	1000-1009	tabella indice def. tasti funzione
4106-4351	100A-10FF	area per definizione tasti funzione
4566-4586	11D6-11EA	area per copia dei primi 21 registri del chip 8564
4616	1208	numero errore
4617-4618	1209-120A	numero linea dell'errore
4624-4625	1210-1211	punt. fine programma BASIC, banco 0
4626-4627	1212-1213	punt. fine memoria per programma, banco 0
4632-4634	1218-121A	JMP + indirizzo salto per USR

ind. dec.	ind. esad.	contenuti
1024-65279	0400-FEFF	MEMORIA VARIABILI PROGRAMMA BASIC NEL BANCO 1

ind. dec.	ind. esad.	contenuti
7168-65279	1C00-FEFF	MEMORIA PER TESTO PROGRAMMA BASIC NEL BANCO 0

ind. dec.	ind. esad.	contenuti
7168-16383	1C00-3FFF	MEMORIA VIDEO ALTA RISOLUZIONE NEL BANCO 0

7168-8191	1C00-1FFF	memoria colore per video a punti

8192-16383	2000-3FFF	mappa video a punti

ind. dec.	ind. esad.	contenuti
16384-45055	4000-AFFF	ROM BASIC
16384	4000	vettore partenza a freddo BASIC
16387	4003	vettore partenza a caldo BASIC
16390	4006	vettore IRQ BASIC

Tabella per i salti alle routine BASIC, il primo byte contiene $4C, codice di JMP, e gli altri due l'indirizzo di inizio della routine, che riportiamo tra parentesi.

44800	AF00	AYINT, conversione floating/intero (33972 $84B4)
44803	AF03	GIVAYF, conversione intero/floating (31036 $793C)
44806	AF06	FOUT, conversione floating/stringa (36418 $8E42)
44809	AF09	VAL1, conversioni stringa/floating (32850 $8052)
44812	AF0C	GETADR, conversione floating/indirizzo (34837 $8815)
44815	AF0F	FLOATC, conversione indirizzo/floating (35957 $8C75)
44818	AF12	FSUB, sottrae FAC1 da memoria (34862 $882E)
44821	AF15	FSUBT, sottrae FAC1 da FAC2 (34865 $8831)
44824	AF18	FADD, somma a FAC1 (34885 $8845)
44827	AF1B	FADDT, somma FAC1 a FAC2 (34888 $8848)
44830	AF1E	FMULT, moltiplica per FAC1 (35364 $8A24)
44833	AF21	FMULTT, moltiplica FAC1 per FAC2 (35367 $8A27)
44836	AF24	FDIV, divide per FAC1 (35657 $8B49)
44839	AF27	FDIVT, divide FAC1 per FAC2 (35660 $8B4C)
44842	AF2A	LOG, calcola logaritmo naturale di FAC1 (35274 $89CA)
44845	AF2D	INT, prende la parte intera di FAC1 (36091 $8CFB)
44848	AF30	SQR, calcola la radice quadrata di FAC1 (36791 $8FB7)
44851	AF33	NEGOP, nega FAC1 (36858 $8FFA)
44854	AF36	FPWR, eleva FAC2 a potenza (36798 $8FBE)
44857	AF39	FPWRT, eleva FAC2 a potenza FAC1 (36801 $8FC1)
44860	AF3C	EXP, calcola EXP di FAC1 (36915 $9033)
44863	AF3F	COS, calcola coseno di FAC1 (37897 $9409)
44866	AF42	SIN, calcola seno di FAC1 (37904 $9410)
44869	AF45	TAN, calcola tangente di FAC1 (37977 $9459)
44872	AF48	ATN, calcola arcotangente di FAC1 (38067 $94B3)
44875	AF4B	ROUND, arrotonda FAC1 (35911 $8C47)
44878	AF4E	ABS, prende valore assoluto di FAC1 (35972 $8C84)

44881	AF51	SIGN, controlla segno di FAC1 (35927 $8C57)
44884	AF54	FCOMP, confronta con FAC1 (35975 $8C87)
44887	AF57	RND0, genera un numero floating a caso (33847 $8437)
44890	AF5A	CONUPK, trasferisce da RAM in FAC2 (35508 $8AB4)
44893	AF5D	ROMUPK, trasferisce da ROM in FAC2 (35465 $8A89)
44896	AF60	MOVFRM, trasferisce da RAM in FAC1 (31365 $7A85)
44899	AF63	MOVFM, trasferisce da ROM in FAC1 (35796 $8BD4)
44902	AF66	MOVMF, trasferisce FAC1 in memoria (35840 $8C00)
44905	AF69	MOVFA, trasferisce FAC2 in FAC1 (35880 $8C28)
44908	AF6C	MOVAF, trasferisce FAC1 in FAC2 (35896 $8C38)
44911	AF6F	OPTAB, ricerca operatore in tabella (18472 $4828)
44914	AF72	DRAWLN, traccia linea (39728 $9B30)
44917	AF75	GPLOT, disegna un punto (39931 $9BFB)
44920	AF78	CIRSUB, traccia un cerchio (26448 $6750)
44923	AF7B	RUN, esegue RUN (23195 $5A9B)
44926	AF7E	RUNC, riporta il puntatore a inizio programma (20979 $51F3)
44929	AF81	CLEAR, esegue CLR (20984 $51F8)
44932	AF84	NEW, esegue NEW (20950 $51D6)
44935	AF87	LNKPRG, sistema linee programma (20303 $4F4F)
44938	AF8A	CRUNCH, trasforma in token (17162 $430A)
44941	AF8D	FNDLIN, ricerca un numero di linea (20580 $5064)
44944	AF90	NEWSTT, prepara la linea successiva (19190 $4AF6)
44947	AF93	EVAL, conversione stringa/floating (30935 $78D7)
44950	AF96	FRMEVL, valuta un'espressione (30703 $77EF)
44953	AF99	NEWSTT (23206 $5AA6)
44956	AF9C	SETEXC (23169 $5A81)
44959	AF9F	LINGET, conversione stringa/numero linea (20640 $50A0)
44962	AFA2	GARBA2, sistema area dinamica stringhe (37610 $92EA)
44965	AFA5	(19917 $4DCD)

ind. dec.	ind. esad.	contenuti
45056-49151	B000-BFFF	ROM MONITOR LINGUAGGIO MACCHINA
45056	B000	vett. part. a freddo MONITOR (JMP $B021)
45059	B003	vett. BRK MONITOR (JMP $B009)

ind. dec.	ind. esad.	contenuti
49152-53247	C000-CFFF	ROM SCREEN EDITOR

Tabella per i salti alle routine, il primo byte contiene $4C, codice di JMP, gli altri due l'indirizzo di inizio della routine, che riportiamo tra parentesi.

49152	C000	inizializza Editor e video (49275 $C07B)
49155	C003	visualizza un carattere (52276 $CC34)
49158	C006	legge un carattere dal buffer tastiera (49716 $C234)
49161	C009	legge un caratt. dal video (49819 $C29B)
49164	C00C	visualizza un carattere sul video (50989 $C72D)
49167	C00F	fornisce numero riga e colonna nella finestra video attuale (52315 $CC5B)
49170	C012	scandisce la tastiera per trovare i tasti premuti (50525 $C55D)
49173	C015	controlla se ripetere un tasto (50769 $C651)
49176	C018	legge o predispone posizione cursore (52330 $CC6A)
49179	C01B	sposta il cursore nel video a 80 col. (52567 $CD57)
49182	C01E	tratta la sequenza con ESC (51649 $C9C1)
49185	C021	definisce un tasto programmabile (52386 $CCA2)
49188	C024	vettore IRQ EDITOR (49556 $C194)
49191	C027	inizializza il set di caratteri per il video a 80 colonne (52748 $CE0C)
49194	C02A	passa da 40 a 80 colonne (52526 $CD2E)
49197	C02D	predispone i limiti di una finestra (51739 $CA1B)

ind. dec.	ind. esad.	contenuti
53248-57343	D000-DFFF	ROM CARATTERI
53248-54271	D000-D3FF	set maiuscolo/grafico normale
54272-55295	D400-D7FF	set maiuscolo/grafico inverso
55296-56319	D800-DBFF	set minuscolo/maiuscolo normale
56320-57343	DC00-DFFF	set minuscolo/maiuscolo inverso

ind. dec.	ind. esad.	contenuti
53248-57343	D000-DFFF	BLOCCO I/O
53248-53296	D000-D030	chip 8564 VIC video a 40 colonne
53280	D020	colore bordo nei bit 3/0
53281	D021	colore sfondo nei bit 3/0
53296	D030	0 per funzionamento SLOW, 1 per funzionamento FAST
54272-54300	D400-D41C	chip SID suono
54528-54539	D500-D50B	chip MMU gestione memoria
54784-54785	D600-D601	chip 8563 video a 80 colonne
55296-56319	D800-DBFF	mappa colore video a 40 colonne
56320-56335	DC00-DC0F	chip CIA 1 input/output

56576-56591	DD00-DD0F	chip CIA 2 input/output
56832-57087	DE00-DEFF	riservato per I/O future espansioni
57088-57098	DF00-DF0A	riservato per I/O future espansioni

ind. dec.	ind. esad.	contenuti
57344-65535	E000-FFFF	ROM KERNAL

Tabella nuovi salti per COMMODORE 128

65351	FF47	SPIN-SPOUT, se carry=0 seleziona SPINP, se carry=1 seleziona SPOUT, per abilitare comunicazione FAST con unità 1571 per input e output
65354	FF4A	CLOSE-ALL, chiude tutti i file aperti sulla periferica il cui DN si trova in A
65357	FF4D	C64MODE, passa al modo 64
65360	FF50	DMA-CALL, comando a periferica per comunicare con cartridge di espansione RAM
65363	FF53	BOOT-CALL, preleva e avvia programma dal disco preparato con settore per autoboot
65366	FF56	PHOENIX, inizializza un cartridge ROM
65369	FF59	LKUPLA, cerca un numero logico di file nella tabella prelevandolo da A; al ritorno LFN, DN e SA sono in A, X e Y se il file esiste
65372	FF5C	LKUPSA, cerca un indirizzo secondario nella tabella prelevandolo da Y, al ritorno dati come per LKUPLA
65375	FF5F	SWAPPER, passa da 40 a 80 colonne; se attivo 80 colonne il bit 7 della locazione 215 ($D7) è a 1
65378	FF62	DLCHR, inizializza il set di caratteri per video a 80 colonne ricopiandolo dalla ROM nella RAM del chip 8563 da $2000 a $3FFF
65381	FF65	PFKEY, assegna una stringa a un tasto funzione
65384	FF68	SETBNK, predispone un banco per operazioni I/O, chiamare prima SETNAM, porre in X il numero del banco che contiene il nome del file e in A il numero del banco che contiene i dati

65387	FF6B	GETCFG, preleva dal registro X il numero del banco da selezionare, da 0 a 15, e ritorna in A il valore necessario per configurare il banco, che deve essere memorizzato in $FF00 con STA $FF00
65390	FF6E	JSRFAR, salta a una routine in qualunque banco con ritorno, prima di chiamarla vanno preparati i byte da 2 a 8 della pagina zero, al ritorno vanno ricaricati i registri corrispondenti prelevando i dati dai byte da 5 a 8 della pagina zero
65393	FF71	JMPFAR, salta a una routine in qualunque banco senza ritorno, prima di chiamarla vanno preparati i byte da 2 a 8 della pagina zero
65396	FF74	INDFET, carica un byte da qualunque banco utilizzando il registro Y
65399	FF77	INDSTA, memorizza un byte in qualunque banco utilizzando il registro Y
65402	FF7A	INDCMP, confronta un byte in qualunque banco utilizzando il registro Y
65405	FF7D	PRIMM, stampa la stringa che sta dopo JSR a questa routine, tale stringa deve terminare con un byte a zero, poi continua con l'istruzione successiva

Tabella salti utilizzati anche sulle altre macchine COMMODORE

65408	FF80	numero versione KERNAL
65409	FF81	CINT, deve essere preceduta dall'istruzione SEI, inizializza i vettori dei salti per le routine dell'EDITOR, utilizzate da IRQ; l'inizializzazione è completa se il bit 6 della locazione 2564 ($0A04) è a 0, altrimenti non inizializza la tastiera
65412	FF84	IOINIT, deve essere preceduta dall'istruzione SEI, inizializza i vettori dei salti per le periferiche, utilizzati da IRQ; se il bit 7 della locazione 2564 ($0A04) è a 0 l'inizializzazione è completa, altrimenti non inizializza il set di caratteri per le 80 colonne
65415	FF87	RAMTAS, inizializza RAM e buffer
65418	FF8A	RESTOR, ripristina valori iniziali per i vettori

		dei salti indiretti delle routine KERNAL, deve essere preceduta dall'istruzione SEI
65421	FF8D	VECTOR, predispone (carry=0) o copia (carry=1) i vettori dei salti indiretti
65424	FF90	SETMSG, abilita (se bit 6 di A a 1) i messaggi di errore delle routine KERNAL e (se bit 7 di A a 1) quelli di controllo
65427	FF93	SECND, invia SA che preleva da A, chiamare prima LISTN
65430	FF96	TKSA, invia SA, che preleva da A
65433	FF99	MEMTOP, predispone o legge il limite superiore della RAM con le modalità di MEMBOT
65436	FF9C	MEMBOT, predispone (carry=0) o legge (carry=1) il limite inferiore della RAM, l'indirizzo in X e Y
65439	FF9F	KEY, legge dalla tastiera, cioè se è stato premuto un tasto pone il codice nel buffer
65442	FFA2	SETTMO, abilita o disabilita IEEE timeout, non usato in modo 128
65445	FFA5	ACPTR, legge un byte dal bus seriale, chiamare prima TALK e, se necessario indirizzo secondario anche TKSA; modifica A
65448	FFA8	CIOUT, scrive un byte sul bus seriale, deve essere preceduta da LISTN, e se è necessario l'indirizzo secondario anche da SECND, il byte viene prelevato da A
65451	FFAB	UNTLK, invia comando UNTALK a una periferica seriale per sospendere invio dati
65454	FFAE	UNLSN, invia comando UNLISTEN a una periferica seriale per sospendere lettura dati
65457	FFB1	LISTN, invia comando LISTEN a una periferica seriale il cui DN si trova in A
65460	FFB4	TALK, invia comando TALK a una periferica seriale il cui DN si trova in A
65463	FFB7	READSS, legge lo stato dell'ultima operazione di I/O in A
65466	FFBA	SETLFS, predispone LFN, DN e SA, che de-

		vono essere in A, X e Y
65469	FFBD	SETNAM, predispone lunghezza e indirizzo del nome del file attuale, prelevando da A la lunghezza e da X e Y l'indirizzo del nome, chiamare prima SETBNK
65472	FFC0	OPEN, apre un file logico, chiamare prima SETNAM, SETBNK e SETLFS
65475	FFC3	CLOSE, chiude un file logico il cui FN si trova in A
65478	FFC6	CHKIN, predispone un canale per input, deve essere preceduta da OPEN e lo LFN deve stare in A
65481	FFC9	CKOUT, predispone un canale per output, deve essere preceduta da OPEN, lo LFN deve essere nel registro X
65484	FFCC	CLRCH, azzera tutti i canali aperti e ripristina i valori di default per tastiera e video, se alcuni canali sono di tipo seriale vanno usate prima UNTLK per quelli di input e UNLSN per quelli di output
65487	FFCF	BASIN, legge un byte da un canale di input e lo pone in A, per default legge dalla tastiera, per altra periferica chiamare prima CHKIN
65490	FFD2	BSOUT, invia un byte a un canale di output prelevandolo da A, per default lavora con il video, se si vuole un'altra periferica chiamare prima CKOUT
65493	FFD5	LOAD, carica e verifica dati da una periferica, prima utilizzare SETNAM, SETBNK e SETLFS, porre 0 in A per caricare e un numero diverso da 0 per verificare; se SA=0 X e Y devono contenere l'indirizzo di inizio
65496	FFD8	SAVE, invia il contenuto della memoria a una periferica, chiamare prima SETNAM, SETBNK e SETLFS
65499	FFDB	SETTIM, predispone il clock prelevando i tre byte da A, X e Y
65502	FFDE	RDTIM, legge il clock in A, X e Y, LOW byte in A
65505	FFE1	STOP, legge il tasto RUN/STOP e se premuto

		chiama CLRCH
65508	FFE4	GETIN, legge un byte dal buffer di input, che per default è la tastiera, per cambiare periferica usare prima CHKIN
65511	FFE7	CLALL, chiude tutti i file e i canali, resetta il puntatore alla tabella dei file (152 $0098)
65514	FFEA	UDTIM, aggiorna il clock, è chiamata normalmente da IRQ. Viene incrementato TIME agli indirizzi 160-162 ($00A0-$00A2) e decrementa TIMER agli indirizzi 2589-2591 ($0A1D-$0A1F)
65517	FFED	SCRORG, legge le dimensioni della finestra video attuale ponendo in A la larghezza del video, in X la larghezza della finestra e in Y la sua altezza
65520	FFF0	PLOT, predispone (carry=0) o legge (carry=1) la posizione del cursore, riga e colonna in X e Y, posizione relativa per la finestra corrente
65523	FFF3	IOBASE, legge una locazione dal blocco di I/O ponendo il byte LOW in X e il byte HIGH in Y
ind. dec.	ind. esad.	contenuti 65280-65535 FF00-FFFF RAM ALTA COMUNE
65280-65284	FF00-FF04	registri comuni MMU
65285-65302	FF05-FF16	banco 1: routine NMI
65303-65330	FF17-FF32	banco 1: routine IRQ
65331-65340	FF33-FF3C	banco 1: routine ritorno da interrupt
65341-65348	FF3D-FF44	banco 1: routine reset
65488-65502	FFD0-FFDE	banco 0: routine cede controllo Z80
65504-65518	FFE0-FFEE	banco 0: routine ritorna controllo 8502
65528-65529	FFF8-FFF9	banco 1: vettore routine reset

REGISTRI CHIP VIC 8564

num. reg.	indirizzo dec.	esad.	contenuti
0	53248	D000	sprite 0, coordinata X (reg. ombra 4566)
1	53249	D001	sprite 0, coordinata Y (reg. ombra 4567)
2	53250	D002	sprite 1, coordinata X (reg. ombra 4568)
3	53251	D003	sprite 1, coordinata Y (reg. ombra 4569)
4	53252	D004	sprite 2, coordinata X (reg. ombra 4570)
5	53253	D005	sprite 2, coordinata Y (reg. ombra 4571)
6	53254	D006	sprite 3, coordinata X (reg. ombra 4572)
7	53255	D007	sprite 3, coordinata Y (reg. ombra 4573)

8	53256	D008	sprite 4, coordinata X (reg. ombra 4574)
9	53257	D009	sprite 4, coordinata Y (reg. ombra 4575)
10	53258	D00A	sprite 5, coordinata X (reg. ombra 4576)
11	53259	D00B	sprite 5, coordinata Y (reg. ombra 4577)
12	53260	D00C	sprite 6, coordinata X (reg. ombra 4578)
13	53261	D00D	sprite 6, coordinata Y (reg. ombra 4579)
14	53262	D00E	sprite 7, coordinata X (reg. ombra 4580)
15	53263	D00F	sprite 7, coordinata Y (reg. ombra 4581)
16	53264	D010	gli 8 bit sono ognuno il bit più significativo della coordinata X degli sprite: il bit di posizione 0 per lo sprite 0, il bit di posizione 7 per lo sprite 7 (reg. ombra 4582)
17	53265	D011	(reg. ombra 4583) bit 7: bit più significativo registro linea bit 6: a 1 per sfondo programmabile bit 5: a 1 per pagina grafica bit 4: a 0 per annullamento schermo bit 3: a 0 per schermo a 24 righe bit 2-0: pos. Y delle scritte (scorrimento fine)
18	53266	D012	registro di linea (reg. ombra 4584)
19	53267	D013	coordinata X penna ottica (reg. ombra 4585)
20	53268	D014	coordinata Y penna ottica (reg. ombra 4586)
21	53269	D015	ogni bit a 1 abilita lo sprite corrispondente: bit 7 corrisponde a sprite 7 bit 0 corrisponde a sprite 0
22	53270	D016	bit 7-6: non usati bit 5: reset bit 4: a 1 per modo multicolore bit 3: a 0 per schermo a 38 colonne bit 2-0: coordinata X delle scritte (scorrimento fine)
23	53271	D017	ogni bit a 1 abilita l'espansione verticale dello sprite corrispondente: bit 7 per lo sprite 7, bit 0 per lo sprite 0
24	53272	D018	bit 7-4: ind. base mappa video bit 3-1: ind. base mappa caratteri bit 0: non usato
25	53273	D019	registro di stato interrupt: bit 7: a 1 da chip 8564 bit 6-4: non usati bit 3: a 1 da penna ottica bit 2: a 1 per collisione sprite tra loro bit 1: a 1 per collisione tra sprite e sfondo bit 0: a 1 da registro di linea
26	53274	D01A	registro per maschera abilitazione interrupt con le stesse corrispondenze del precedente il bit a 1 significa abilitato
27	53275	D01B	registro priorità sprite-sfondo bit a 0 per sprite a 1 per sfondo il bit 7 corrisponde allo sprite 7 il bit 0 corrisponde allo sprite 0
28	53276	D01C	multicolore per gli sprite: 0 per no, 1 per si

			il bit 7 corrisponde allo sprite 7 il bit 0 corrisponde allo sprite 0
29	53277	D01D	ogni bit a 1 abilita l'espansione orizzontale dello sprite corrispondente: bit 7 per lo sprite 7, bit 0 per lo sprite 0
30	53278	D01E	ogni bit a 1 corrisponde alla collisione di uno sprite con un altro sprite: bit 7 per lo sprite 7, bit 0 per lo sprite 0
31	53279	D01F	ogni bit a 1 corrisponde alla collisione di uno sprite con lo sfondo: bit 7 per lo sprite 7, bit 0 per lo sprite 0
32	53280	D020	colore del bordo nei bit 3-0
33	53281	D021	colore di sfondo (in modo testo) nei bit 3-0
34	53282	D022	colore 1 nei bit 3-0
35	53283	D023	colore 2 nei bit 3-0
36	53284	D024	colore 3 nei bit 3-0
37	53285	D025	colore 2 sprite multicolore nei bit 3-0
38	53286	D026	colore 3 sprite multicolore nei bit 3-0
39	53287	D027	colore sprite 0 nei bit 3-0
40	53288	D028	colore sprite 1 nei bit 3-0
41	53289	D029	colore sprite 2 nei bit 3-0
42	53290	D02A	colore sprite 3 nei bit 3-0
43	53291	D02B	colore sprite 4 nei bit 3-0
44	53292	D02C	colore sprite 5 nei bit 3-0
45	53293	D02D	colore sprite 6 nei bit 3-0
46	53294	D02E	colore sprite 7 nei bit 3-0
47	53295	D02F	bit 3-0: scansione tastiera
48	53296	D030	bit 1-0: 0 per SLOW, 1 per FAST

REGISTRI CHIP 6581 SID

num. reg.	indirizzo dec.	esad.	contenuti
0	54272	D400	LO byte frequenza voce 1
1	54273	D401	HI byte frequenza voce 1
2	54274	D402	LO byte larghezza onda impulsiva (rettangolare) voce 1
3	54275	D403	bit 7-4: non usati bit 3-0: semibyte HI larghezza onda impulsiva voce 1
4	54276	D404	registro controllo voce 1 bit 7: a 1 forma d'onda casuale (rumore) bit 6: a 1 forma d'onda impulsiva bit 5: a 1 forma d'onda a dente di sega bit 4: a 1 forma d'onda triangolare bit 3: a 1 disabilita oscillatore 1 bit 2: a 1 oscillatore 1 modulato con uscita oscillatore 3 bit 1: a 1 oscillatore 1 sincronizzato con frequenza oscillatore 3 bit 0: a 1 per partenza att./dec./sus. a 0 per partenza release
5	54277	D405	bit 7-4: attack generatore 1 bit 3-0: decay generatore 1

6	54278	D406	bit 7-4: sustain generatore 1 bit 3-0: release generatore 1
7	54279	D407	LO byte frequenza voce 2
8	54280	D408	HI byte frequenza voce 2
9	54281	D409	LO byte larghezza onda impulsiva voce 2
10	54282	D40A	bit 7-4: non usati bit 3-0: semibyte HI larghezza onda impulsiva voce 2
11	54283	D40B	registro controllo voce 2 bit 7: a 1 forma d'onda casuale (rumore) bit 6: a 1 forma d'onda impulsiva bit 5: a 1 forma d'onda a dente di sega bit 4: a 1 forma d'onda triangolare bit 3: a 1 disabilita oscillatore 2 bit 2: a 1 oscillatore 2 modulato con uscita oscillatore 1 bit 1: a 1 oscillatore 2 sincronizzato con frequenza oscillatore 1 bit 0: a 1 per partenza att./dec./sus. a 0 per partenza release
12	54284	D40C	bit 7-4: attack generatore 2 bit 3-0: decay generatore 2
13	54285	D40D	bit 7-4: sustain generatore 2 bit 3-0: release generatore 2
14	54286	D40E	LO byte frequenza voce 3
15	54287	D40F	HI byte frequenza voce 3
16	54288	D410	LO byte larghezza onda impulsiva voce 3
17	54289	D411	bit 7-4: non usati bit 3-0: semibyte HI larghezza onda impulsiva voce 3
18	54290	D412	registro controllo voce 3 bit 7: a 1 forma d'onda casuale (rumore) bit 6: a 1 forma d'onda impulsiva bit 5: a 1 forma d'onda a dente di sega bit 4: a 1 forma d'onda triangolare bit 3: a 1 disabilita oscillatore 3 bit 2: a 1 oscillatore 3 modulato con uscita oscillatore 2 bit 1: a 1 oscillatore 3 sincronizzato con frequenza oscillatore 2 bit 0: a 1 per partenza att./dec./sus. a 0 per partenza release
19	54291	D413	bit 7-4: attack generatore 3 bit 3-0: decay generatore 3
20	54291	D414	bit 7-4: sustain generatore 3 bit 3-0: release generatore 3
21	54292	D415	bit 7-3: non usati bit 2-0: bit meno significativi della frequenza di taglio del filtro
22	54293	D416	HI byte della frequenza di taglio del filtro
23	54294	D417	bit 7-4: risonanza filtro bit 3: filtra input esterno bit 2: filtra voce 3

			bit 1: filtra voce 2
			bit 0: filtra voce 1
24	54295	D418	bit 7: a 0 disabilita uscita voce 3
			bit 6: a 1 filtro passa alto
			bit 5: a 1 filtro passa banda
			bit 4: a 1 filtro passa basso
			bit 3-0: volume
25	54296	D419	convertitore analogico digitale paddle (pot X)
26	54297	D41A	convertitore analogico digitale paddle (pot Y)
27	54298	D41B	uscita voce 3
28	54299	D41C	uscita inviluppo voce 3

Nota: i primi 25 registri sono a sola scrittura e gli ultimi 4 a sola lettura.

REGISTRI CHIP 8563

indirizzo dec.	esad.	contenuti
54784	D600	intermediario per uno dei 37 registri del chip ponendo un numero da 0 a 36 in questo registro quando i bit 7 e 6 vanno a 1 si puó leggere dal registro successivo il contenuto o scrivervi il nuovo contenuto per il registro selezionato bit 7: stato registro bit 6: penna ottica bit 4-0: indirizzo
54785	D601	immagine del registro selezionato (dato)

Contenuti dei 37 registri accessibili mediante 54784 e 54785 ($D600-$D601)

reg.	contenuti
0	totale orizzontale, numero dei caratteri tra i sincronismi orizzontali successivi meno uno
1	visualizzato orizzontale, numero dei caratteri visualizzati per ogni riga
2	sincronizzazione posizione orizzontale, numero di caratteri dall'inizio della riga visualizzata al prossimo sincronismo orizzontale
3	bit 7-4: ampiezza sincronizzazione verticale, larghezza in numero di linee di scansione dell'impulso di sincronismo verticale bit 3-0: ampiezza sincronizzazione orizzontale
4	totale verticale, numero di righe per frame meno uno, si veda anche registro 5
5	bit 7-5: non usati bit 4-0: aggiustamento totale verticale, numero di linee di scansione aggiunte al registro 4 per una regolazione fine della velocità di rinfresco
6	visualizzato verticale, numero di righe visualizzate
7	sincronizzazione posizione verticale, numero di righe dalla prima riga visualizzata al prossimo sincronismo verticale più uno
8	bit 7-2: non usati bit 1-0: controllo interlacciamento: 00 o 10, non interlacciato 01, sincronismo interlacciato, dati identici visualizzati su

frame alternati
11, sincronismo interlacciato e video, linee di scansione alternate su frame alternati

9 bit 7-5: non usati
bit 4-0: totale caratteri verticali, numero di linee di scansione per riga meno uno

10 bit 7: non usato
bit 6-5: modo del cursore
00, non lampeggia
01, non appare
10, lampeggia a 1/16 della velocità di rinfresco
11, lampeggia a 1/32 della velocità di rinfresco
bit 4-0: inizio scansione, linea di scansione iniziale del cursore

11 bit 7-5: non usati
bit 4-0: fine scansione, linea di scansione finale del cursore

12 HI byte dell'indirizzo iniziale all'interno dei 16K della memoria RAM del chip della mappa del video (bit 7 e 6 a zero)

13 LO byte dell'indirizzo il cui HI byte sta nel registro 12

14 HI byte dell'indirizzo del cursore nei 16K di memoria RAM del chip

15 LO byte dell'indirizzo il cui HI byte sta nel registro 14

16 posizione verticale penna ottica, numero di caratteri dalla prima riga visualizzata alla posizione della penna ottica più uno

17 posizione orizzontale penna ottica, numero di caratteri dall'inizio della riga alla posizione della penna ottica più otto

18 HI byte dell'indirizzo di memoria, all'interno dei 16K della memoria RAM del chip, dove avrà luogo la prossima operazione di lettura o scrittura

19 LO byte dell'indirizzo il cui HI byte sta nel registro 18; i registri 18 e 19 contengono un indirizzo che si incrementa automaticamente dopo ogni operazione

20 HI byte dell'indirizzo iniziale degli attributi dei caratteri visualizzati

21 LO byte dell'indirizzo il cui HI byte sta nel registro 20

22 bit 7-4: totale del carattere, larghezza del carattere in pixel meno uno, include anche la spaziatura tra i caratteri
bit 3-0: totale visualizzato orizzontale del carattere, larghezza in pixel della parte visualizzata del carattere esclusa la spaziatura tra i caratteri

23 bit 7-5: non usati
bit 4-0: totale visualizzato verticale del carattere, altezza in numero di linee di scansione meno uno, esclusa la spaziatura verticale tra i caratteri

24 bit 7: copia del blocco: 0 per scrivere, 1 per leggere
bit 6: campo inverso: 0 normale, 1 in campo inverso
bit 5: lampeggiamento: 0 per 1/16 velocità di rinfresco, 1 per 1/32 velocità di rinfresco
bit 4-0: scorrimento fine verticale, numero di linee di scansione da spostare verso l'alto

25 bit 7: modo: 0 testo, 1 mappa di byte
bit 6: abilitazione attributi:
0, disabilitato, il colore dei caratteri viene determinato dai bit 7-4 del registro 26
1, abilitato, gli attributi sono contenuti nel byte della mappa

degli attributi
bit 5: modo semigrafico:
0, lo spazio orizzontale di separazione dei caratteri viene visualizzato nel colore dello sfondo
1, lo spazio orizzontale di separazione dei caratteri viene visualizzato nel colore dell'ultima colonna di pixel
bit 4: larghezza doppia pixel:
0, larghezza normale
1, larghezza doppia
bit 3-0: scorrimento fine orizzontale, numero di pixel da spostare a sinistra

26 bit 7-4: colore caratteri
bit 3-0: colore sfondo

27 incremento indirizzo per riga, il valore viene aggiunto automaticamente ai registri 12-13 e 20-21 ad ogni lettura o scrittura del registro 31

28 bit 7-5: indirizzo base caratteri per selezionare il banco in cui risiedono i caratteri
bit 4-0: non usati

29 bit 7-5: non usati
bit 4-0: contatore linee di scansione, numero di linee di scansione per la sottolineatura

30 contatore, numero di cicli dell'operazione di blocco da eseguire, il valore 0 corrisponde a 256

31 dato, viene usato per scrivere o è il risultato di una lettura

32 HI byte dell'indirizzo di inizio del blocco da copiare, l'indirizzo di destinazione sta nei registri 18 e 19

33 LO byte dell'indirizzo il cui byte HI sta nel registro 32

34 abilitazione inizio visualizzazione (non modificare)

35 abilitazione fine visualizzazione (non modificare)

36 bit 7-4: non usati
bit 3-0: velocità di rinfresco (non modificare)

Utilizzo 16K RAM chip 8563

ind. dec.	ind. esad.	contenuti
0-1999	0000-07CF	mappa video 80 colonne (2000 caratteri)
2000-4047	0800-0FCF	attributi caratteri video
8192-16383	2000-3FFF	definizione caratteri

Nota: Pixel significa punto. L'immagine di un byte sul video corrisponde a 8 pixel (o a 8 punti).

MAPPA REGISTRI CHIP 6526 CIA

num.reg.	descrizione
0	PRA, registro dati A
1	PRB, registro dati B
2	DDRA, direzione dati registro A
3	DDRB, direzione dati registro B
4	TA LO, registro LOW timer A
5	TA HI, registro HIGH timer A
6	TB LO, registro LOW timer B

7	TB HI, registro HIGH timer B
8	TOD 10THS, decimi di secondo orologio TOD
9	TOD SEC, secondi orologio TOD
10	TOD MIN, minuti orologio TOD
11	TOD HR, ore orologio TOD
12	SDR, registro seriale dati
13	ICR, registro controllo interrupt
14	CRA, registro di controllo A
15	CRB, registro di controllo B

Registri CIA 1 da 56320 a 56335 ($DC00-$DC0F)

56320 ($DC00) PRA

bit 7-0:	rilevamento colonna tasto premuto (output)
bit 7:	selezione paddle da porta 2
bit 6:	selezione paddle da porta 1
bit 4:	rilevamento pulsante FIRE joystick 2
bit 3-0:	rilevamento posizione joystick 2
bit 3-2:	rilevamento pulsanti FIRE paddle

Nota: l'utilizzo multiplo di alcuni bit puó creare dei problemi di cattiva interpretazione dei segnali, in conseguenza non si devono usare contemporaneamente le periferiche che possono generare contrasti. Questo registro rileva anche i segnali del mouse e della penna ottica.

56321 ($DC01) PRB

bit 7-0:	scansione tastiera per righe (input)
bit 7:	selezione modo timer A
bit 6:	selezione modo timer B
bit 4:	rilevamento pulsante FIRE joystick 1
bit 3-0:	rilevamento posizione joystick 1

Nota: quando si devono utilizzare contemporaneamente joystick e tastiera conviene collegare il joystick alla porta 2.

56324 ($DC04) TA LO Timer A
56325 ($DC05) TA HI
56334 ($DC0E) CRA
sono utilizzati per la gestione veloce dell'interfaccia seriale (disk 1571).

56326 ($DC06) TB LO Timer B
56327 ($DC07) TB HI
56335 ($DC0F) CRB
sono utilizzati per la gestione del registratore COMMODORE.

Registri CIA 2 da 56576 a 56591 ($DD00-$DD0F)

56576 ($DD00) PRA

bit 7:	BUS seriale input DATA (1 bit in input)
bit 6:	BUS seriale input CLK (clock di ricezione)

bit 5:	BUS seriale output DATA (1 bit in output)
bit 4:	BUS seriale output CLK (clock di trasmissione)
bit 3:	BUS seriale output ATN (richiesta comunicazione)
bit 2:	RS-232 output DATA (1 bit di dato in output)
bit 1-0:	blocco di memoria visibile dal chip VIC 8564 00 blocco 3 da 49152 a 65535 ($C000-$FFFF) 01 blocco 2 da 32768 a 49151 ($8000-$BFFF) 10 blocco 1 da 16384 a 32767 ($4000-$7FFF) 11 blocco 0 da 0 a 16383 ($0000-$3FFF)

Nota: utilizzato come per il COMMODORE 64 per la gestione della comunicazione seriale a velocità normale.

56577 ($DD01) PRB

bit 7-0:	un byte I/O user port
bit 7:	RS-232 data set ready
bit 6:	RS-232 clear to send
bit 4:	RS-232 carrier detect
bit 3:	RS-232 ring indicator
bit 2:	RS-232 data terminal ready
bit 1:	RS-232 request to send
bit 0:	RS-232 input DATA (1 bit in input)

Nota: utilizzato sia per la comunicazione tramite user port che per la simulazione della comunicazione con il protocollo RS-232.

56580 ($DD04) TA LO Timer A
56581 ($DD05) TA HI
56590 ($DD0E) CRA
utilizzati per la baud rate del protocollo RS-232

56582 ($DD06) TB LO Timer B
56583 ($DD07) TB HI
56591 ($DD0F) CRB
utilizzati per il controllo della trasmissione RS-232

56589 ($DD0D) ICR
utilizzato per generare NMI per temporizzazione trasmissione RS232

3.3 UTILIZZO DELLA MEMORIA NEI PROGRAMMI IN BASIC

Per sapere dove è localizzato il programma in BASIC possiamo eseguire in modo immediato le seguenti istruzioni:

```
BANK 0:PRINT PEEK(45) + 256 * PEEK(46)
BANK 0:PRINT PEEK(4624) + 256 * PEEK(4625)
```

con le quali, dopo aver attivato il banco 0, leggiamo il puntatore all'inizio (45/46, $2D/$2E) e alla fine (4624/4625, $1210/$1211) del programma BASIC. Di norma

troviamo per il primo l'indirizzo 7169 ($1C01), infatti in 7168 il sistema mette uno zero binario, otto bit zero. Se abbiamo attivato uno dei modi grafici, otteniamo invece 16385 ($4001), infatti a partire da 7168 viene riservato lo spazio per la pagina grafica e la mappa dei colori in grafica.
L'utente puó modificare l'inizio della zona per il programma BASIC, agendo direttamente sui relativi puntatori, e riservare dello spazio o prima o dopo le istruzioni BASIC per altri scopi come, per esempio, la memorizzazione di programmi in linguaggio macchina.

Il sistema memorizza il programma BASIC in un linguaggio intermedio, traducendo le parole chiave e i simboli del linguaggio in codici numerici che occupano uno o due byte ciascuno; scorrendo l'elenco delle routine del sistema si trovano quelle che provvedono alla traduzione.
Per vedere il linguaggio intermedio abbiamo preparato un programma che legge e stampa se stesso. Prima di stendere questo programma abbiamo analizzato in immediato come viene memorizzata una linea, riscontrando i fatti che elenchiamo.
– I primi due byte formano l'indirizzo di LINK, cioè puntano all'inizio della linea seguente. Il primo è il byte LOW, il secondo il byte HIGH. L'ultima linea del programma ha questi due byte a zero.
– I due byte seguenti contengono il numero della linea, nella forma LOW, HIGH.
– Seguono i byte che contengono la linea. Le parole chiave sono espresse comprimendole in uno o due byte (TOKEN) e gli altri elementi carattere per carattere in codice ASCII.
– La linea termina con un byte che contiene zero.
Si noti che quando diciamo che un byte contiene zero, intendiamo che tutti i suoi bit sono a zero (zero binario); la cifra zero è invece espressa dal codice ASCII 48 ($30).
Da quanto esposto vediamo che il carattere RETURN, da noi utilizzato per chiudere una linea di programma, espresso dal codice ASCII 13 ($0D), non viene memorizzato, ma si trasforma in un byte contenete zero.

A questo punto è possibile determinare l'algoritmo da applicare per stendere il nostro programma. Decidiamo di stampare i risultati nella forma:

ind. linea ** 2 byte di link e LINK= ind. linea seguente
NUMERO LINEA: num. linea

byte successivi andando a capo ogni 12 elementi.

Dobbiamo eseguire i seguenti passi:
– 1. aprire la stampante e leggere l'indirizzo di inizio del programma;
– 2. leggere i primi due byte, se contengono zero il programma è finito, quindi andare al punto 7, altrimenti essi rappresentano il LINK;
– 3. stampare la prima riga, calcolare il numero della linea e stampare la seconda riga;
– 4. puntare al quinto byte, dove inizia la prima istruzione della linea, istituire un contatore per contare gli elementi di stampa;
– 5. leggere un byte, se a zero terminare la stampa della linea e tornare al punto 2, dopo aver incrementato il puntatore;
– 6. stampare il byte, incrementare il puntatore e il contatore per la stampa, se questo ha raggiunto 12, andare a capo e azzerarlo, tornare al punto 5;
– 7. chiudere la stampante; il programma è terminato.

Utilizziamo le seguenti variabili:
– X, per l'indirizzo del byte da leggere, parte dall'indirizzo iniziale del programma;
– Y1 e Y2, per leggere i byte;
– C, per contare gli elementi di stampa.
Inoltre utilizziamo la maschera di stampa # # # # per incolonnare i dati.

Segue la lista del programma PRGINMEM realizzato, e, nella figura 3.4 la parte iniziale e finale dei risultati.

```
1 REM PRGINMEM
3 REM TRASFERISCE USCITA SU STAMPANTE
5 OPEN 4,4:CMD 4
7 PRINT"PROGRAMMA IN MEMORIA":PRINT
9 REM CALCOLA INDIRIZZO INIZIO BASIC
11 X=PEEK(45)+256*PEEK(46)
13 REM CONTROLLA SE X E X+1 CONTENGONO 0
15 REM SE SI IL PROGRAMMA E' FINITO
17 Y1=PEEK(X):Y2=PEEK(X+1)
19 IF Y1+Y2=0 THEN PRINT X;"**";Y1;Y2:GOTO 59
21 REM ALTRIMENTI X E X+1 CONTENGONO IL LINK
23 PRINT X;"**";Y1;Y2;"LINK=";Y1+256*Y2
25 REM I DUE BYTE SEGUENTI CONTENGONO
27 REM IL NUMERO DI LINEA
29 Y1=PEEK(X+2):Y2=PEEK(X+3)
31 REM STAMPA IL NUMERO DI LINEA
33 PRINT "NUMERO LINEA: ";Y1+256*Y2
35 REM C CONTA IL NUMERO DEGLI ELEMENTI STAMPATI
37 X=X+4:C=0:REM PUNTA AL QUINTO BYTE
39 REM SE IL BYTE CONTIENE 0 E' FINITA LA LINEA
41 Y1=PEEK(X):IF Y1=0 THEN 53
43 REM STAMPA BYTE DELLA LINEA
45 PRINT USING "####";Y1;:X=X+1:C=C+1
47 REM VA A CAPO DOPO 12 ELEMENTI
49 IF C=12 THEN PRINT:C=0
51 GOTO 41
53 PRINT USING "####";Y1:X=X+1:GOTO 17
55 REM CHIUDE LA STAMPANTE
57 REM E TRASFERISCE L'USCITA AL VIDEO
59 PRINT#4:CLOSE 4:STOP
```

```
PROGRAMMA IN MEMORIA

 7169 ** 16  28 LINK= 7184
NUMERO LINEA:  1
 143  32  80  82  71  73  78  77  69  77   0
 7184 ** 54  28 LINK= 7222
```

```
NUMERO LINEA:  3
 143  32  84  82  65  83  70  69  82  73  83  67
  69  32  85  83  67  73  84  65  32  83  85  32
  83  84  65  77  80  65  78  84  69   0
 7222 ** 68  28 LINK= 7236
NUMERO LINEA:  5
 159  32  52  44  52  58 157  32  52   0
 7236 ** 98  28 LINK= 7266
NUMERO LINEA:  7
 153  34  80  82  79  71  82  65  77  77  65  32
  73  78  32  77  69  77  79  82  73  65  34  58
 153   0
 7266 ** 135  28 LINK= 7303
NUMERO LINEA:  9
 143  32  67  65  76  67  79  76  65  32  73  78
  68  73  82  73  90  90  79  32  73  78  73  90
  73  79  32  66  65  83  73  67   0
 7303 ** 157  28 LINK= 7325
NUMERO LINEA:  11
  88 178 194  40  52  53  41 170  50  53  54 172
 194  40  52  54  41   0
 7325 ** 197  28 LINK= 7365

NUMERO LINEA:  47
 143  32  86  65  32  65  32  67  65  80  79  32
  68  79  80  79  32  49  50  32  69  76  69  77
  69  78  84  73   0
 7940 ** 23  31 LINK= 7959
NUMERO LINEA:  49
 139  32  67 178  49  50  32 167  32 153  58  67
 178  48   0
 7959 ** 32  31 LINK= 7968
NUMERO LINEA:  51
 137  32  52  49   0
 7968 ** 61  31 LINK= 7997
NUMERO LINEA:  53
 153  32 251  32  34  35  35  35  35  34  59  89
  49  58  88 178  88 170  49  58 137  32  49  55
   0
 7997 ** 87  31 LINK= 8023
NUMERO LINEA:  55
 143  32  67  72  73  85  68  69  32  76  65  32
  83  84  65  77  80  65  78  84  69   0
 8023 ** 125  31 LINK= 8061
NUMERO LINEA:  57
 143  32  69  32  84  82  65  83  70  69  82  73
  83  67  69  32  76  39  85  83  67  73  84  65
  32  65  76  32  86  73  68  69  79   0
 8061 ** 138  31 LINK= 8074
```

```
NUMERO LINEA:  59
 152  52  58 160  32  52  58 144   0
 8074 ** 0  0
```

Fig. 3.4 Parte dei risultati del programma PRGINMEM

Si noti che per vedere il linguaggio intermedio si puó utilizzare il MONITOR e chiedere la visualizzazione della zona di memoria che interessa; solo che appare il contenuto dei byte in esadecimale e l'interpretazione risulta più difficile.

Dai risultati del programma PRGINMEM possiamo dedurre i codici di alcune parole chiave. Ci siamo posti il problema di trovare tutte le descrizioni dei TOKEN, sia quelli a uno, che quelli a due byte; essi devono essere memorizzati da qualche parte nella memoria ROM. Una strada da seguire puó essere quella di disassemblare con il MONITOR le routine ad essi relative e cercare in esse gli indirizzi delle tabelle. Noi abbiamo proceduto in altro modo. Abbiamo scritto un programma orribile, cioè di questo tipo:

1 END:FOR:NEXT

e siamo andati a leggere in memoria i TOKEN relativi alle tre parole chiave, trovando 128, 129 e 130, come per il COMMODORE 64. Si noti che il programma, anche se è ovviamente errato, non dà errore se non lo eseguiamo; l'EDITOR non controlla la sintassi delle linee. Per inciso, il programma orribile puó essere continuato con altre linee, elencando tutte le parole chiave, e trovando così tutti i TOKEN, a uno e a due byte. A noi interessava conoscere i TOKEN delle tre parole chiave elencate. Per trovare le tabelle in memoria abbiamo utilizzato il MONITOR, eseguendo in sequenza i comandi di ricerca:

H F0000 FFFFF 'EN
H F0000 FFFFF 'FO
H F0000 FFFFF 'NEX

e ottenendo i risultati:

F020F F0A80 F4417 F45D6 F45E8 per la ricerca di 'EN
F4634 F4650 F4691 F488C F493A
F4A2E F4A60 F4A6C

F020F F0A80 F441A F4878 F48DF per la ricerca di 'FO
F499A F49F1 F4A36 FF6C7 FF714

F020F F0A80 F441D F48D2 per la ricerca di 'NEX

dove la ricerca è stata fatta senza usare l'ultimo carattere dato che non eravamo sicuri sul modo del suo utilizzo.
A questo punto eravamo abbastanza sicuri che la tabella dei TOKEN a un byte doveva trovarsi a partire dal byte di indirizzo $F4417, infatti la sequenza $F4417, $F441A, $F441D si presentava bene.
Abbiamo letto con il MONITOR la memoria a partire dall'indirizzo $F4417 (decimale 17431), ci siamo confermati nell'idea e abbiamo visto che l'ultimo carattere di ogni

parola chiave viene modificato aggiungendo 128 al suo codice ASCII. Facendo un po' di conti, supponendo che il codice TOKEN non potesse superare 255, abbiamo trovato che l'ultimo TOKEN a un byte corrisponde alla parola chiave WHILE ed è 253. A partire dalla parola chiave BANK vengono usati due byte; utilizzando un secondo programma orribile abbiamo trovato il valore dei due byte per le parole chiave non comprese nella prima tabella e in base a tutti questi elementi abbiamo potuto scrivere i due programmi TOKEN1 e TOKEN2 che elencano le parole chiave e i TOKEN corrispondenti.

La tabella di descrizione dei TOKEN a un byte si trova nella ROM da 17431 a 17927 e i valori dei TOKEN vanno da 128 a 253. Per stampare tale tabella dobbiamo procedere così:
- 1. aprire la stampante;
- 2. inizializzare una variabile a 128;
- 3. inviare il codice di controllo per l'ingrandimento dei caratteri, stampare l'intestazione e porre a stringa nulla una variabile dove memorizzare la descrizione;
- 4. leggere in ciclo i byte dall'indirizzo 17431 all'indirizzo 17927;
- 5. se il numero è minore o uguale a 127 aggiungere il carattere corrispondente alla stringa e tornare al punto 4;
- 6. altrimenti sottrarre 128 al numero, aggiungere il carattere corrispondente alla stringa, stampare il codice e la stringa, incrementare il codice di 1, porre a stringa nulla la variabile della descrizione e tornare al punto 4;
- 7. inviare alla stampante il carattere di controllo per tornare alla stampa normale e chiudere la stampante.

Abbiamo utilizzato le variabili:
- J, per i codici TOKEN;
- K, per il controllo del ciclo;
- L$, per la descrizione.
- Ci siamo serviti di maschere di stampa per incolonnare i risultati.

Segue il programma TOKEN1 e nella figura 3.5 sono riportati i suoi risultati, ponendoli su due colonne.

```
 T O K E N

COD.  DES.

128     END
129     FOR
130     NEXT
131     DATA
132     INPUT#
133     INPUT
134     DIM
135     READ
136     LET
137     GOTO
138     RUN
139     IF
140     RESTORE
141     GOSUB
142     RETURN
143     REM
144     STOP
145     ON
146     WAIT
147     LOAD
148     SAVE
149     VERIFY
150     DEF
151     POKE
152     PRINT#
153     PRINT
154     CONT
155     LIST
156     CLR
157     CMD
158     SYS
159     OPEN
160     CLOSE
161     GET
```

```
162     NEW          208     RDOT
163     TAB<         209     DEC
164     TO           210     HEX$
165     FN           211     ERR$
166     SPC(         212     INSTR
167     THEN         213     ELSE
168     NOT          214     RESUME
169     STEP         215     TRAP
170     +            216     TRON
171     -            217     TROFF
172     *            218     SOUND
173     /            219     VOL
174     ↑            220     AUTO
175     AND          221     PUDEF
176     OR           222     GRAPHIC
177     >            223     PAINT
178     =            224     CHAR
179     <            225     BOX
180     SGN          226     CIRCLE
181     INT          227     GSHAPE
182     ABS          228     SSHAPE
183     USR          229     DRAW
184     FRE          230     LOCATE
185     POS          231     COLOR
186     SQR          232     SCNCLR
187     RND          233     SCALE
188     LOG          234     HELP
189     EXP          235     DO
190     COS          236     LOOP
191     SIN          237     EXIT
192     TAN          238     DIRECTORY
193     ATN          239     DSAVE
194     PEEK         240     DLOAD
195     LEN          241     HEADER
196     STR$         242     SCRATCH
197     VAL          243     COLLECT
198     ASC          244     COPY
199     CHR$         245     RENAME
200     LEFT$        246     BACKUP
201     RIGHT$       247     DELETE
202     MID$         248     RENUMBER
203     GO           249     KEY
204     RGR          250     MONITOR
205     RCLR         251     USING
206                  252     UNTIL
207     JOY          253     WHILE
```

Fig. 3.5 Risultati del programma TOKEN1

```
1 REM TOKEN1
3 REM TOKEN A UN BYTE
5 REM OGNI PAROLA CHIAVE O SIMBOLO TERMINA
7 REM CON UN CODICE MAGGIORE DI 127
9 OPEN 4,4:CMD 4:REM APRE STAMPANTE
```

```
11 J=128:REM CODICE PARTE DA 128
13 PRINT CHR$(14);"  T O K E N":PRINT
15 PRINT " COD. DES.":PRINT:L$=""
17 FOR K=17431 TO 17927:REM LIMITI TABELLA
19 BANK 15:L=PEEK(K)
21 IF L<=127 THEN L$=L$+CHR$(L):NEXT K
23 L=L-128:L$=L$+CHR$(L)
25 PRINT USING "####   ";J;
27 PRINT USING "################";L$
29 J=J+1:L$="":NEXT K:PRINT CHR$(15)
31 REM CHIUDE STAMPANTE
33 PRINT#4:CLOSE 4:STOP
```

Il programma TOKEN2 è abbastanza simile al precedente, ma, non avendo studiato l'algoritmo di calcolo dei TOKEN a due byte, li abbiamo trovati utilizzando un programma orribile e leggendoli dalla memoria con il MONITOR; i valori sono stati introdotti nel programma in esadecimale con delle linee DATA e letti come stringhe. Utilizziamo due variabili in più, X$ e Y$, per leggere i due byte esadecimali con l'istruzione READ, e una variabile in meno, la J, che non serve più. La tabella di descrizione si trova dal byte di indirizzo 17929 a quello di indirizzo 18168. Seguono il programma TOKEN2 e, nella figura 3.6, i suoi risultati.

```
1 REM TOKEN2
3 REM TOKEN A DUE BYTE
5 REM OGNI PAROLA CHIAVE O SIMBOLO TERMINA
7 REM CON UN CODICE MAGGIORE DI 127
9 OPEN 4,4:CMD 4:REM APRE STAMPANTE
11 REM STAMPA I DUE BYTE DEL CODICE ESADECIMALE
13 PRINT CHR$(14);"  T O K E N":PRINT
15 PRINT " COD. DES.":PRINT:L$=""
17 FOR K=17929 TO 18168:REM LIMITI TABELLA
19 BANK 15:L=PEEK(K)
21 IF L<=127 THEN L$=L$+CHR$(L):NEXT K
23 L=L-128:L$=L$+CHR$(L)
25 READ X$:READ Y$:REM LEGGE CODICI ESADECIMALI
27 PRINT USING "## ";X$,Y$;
29 PRINT USING "################";L$
31 L$="":NEXT K:PRINT CHR$(15)
33 REM CHIUDE STAMPANTE
35 PRINT#4:CLOSE 4:STOP
37 DATA FE,02,FE,03,FE,04,FE,05,FE,06,FE,07
39 DATA FE,08,FE,09,FE,0A,FE,0B,FE,0C,FE,0D
41 DATA FE,0E,FE,0F,FE,10,FE,11,FE,12,FE,13
43 DATA FE,14,FE,15,FE,16,FE,17,FE,18,FE,19
45 DATA FE,1A,FE,1B,FE,1C,FE,1D,FE,1E,FE,1F
47 DATA FE,20,FE,21,FE,22,FE,23,FE,24,FE,25
49 DATA FE,26,CE,02,CE,03,CE,04,CE,05,CE,06
51 DATA CE,07,CE,08,CE,09,CE,0A
```

```
  T O K E N
 COD.  DES.

FE  02  BANK
FE  03  FILTER
FE  04  PLAY
FE  05  TEMPO
FE  06  MOVSPR
FE  07  SPRITE
FE  08  SPRCOLOR
FE  09  RREG
FE  0A  ENVELOPE
FE  0B  SLEEP
FE  0C  CATALOG
FE  0D  DOPEN
FE  0E  APPEND
FE  0F  DCLOSE
FE  10  BSAVE
FE  11  BLOAD
FE  12  RECORD
FE  13  CONCAT
FE  14  DVERIFY
FE  15  DCLEAR
FE  16  SPRSAV
FE  17  COLLISION
FE  18  BEGIN
FE  19  BEND
FE  1A  WINDOW
FE  1B  BOOT
FE  1C  WIDTH
FE  1D  SPRDEF
FE  1E  QUIT
FE  1F  STASH
FE  20
FE  21  FETCH
FE  22
FE  23  SWAP
FE  24  OFF
FE  25  FAST
FE  26  SLOW
CE  02  POT
CE  03  BUMP
CE  04  PEN
CE  05  RSPPOS
CE  06  RSPRITE
CE  07  RSPCOLOR
CE  08  XOR
CE  09  RWINDOW
CE  0A  POINTER
```

Fig. 3.6 Risultati del programma TOKEN2

Il nome delle variabili viene memorizzato nel programma in BASIC carattere per carattere; il suffisso, se presente, occupa l'ultimo byte. Ci occupiamo ora del modo nel quale le variabili sono memorizzate nella zona di memoria appartenente al banco 1 e riservata ad esse. Si noti che tale zona è molto vasta, per cui è possibile tenere in memoria molti dati. Un'applicazione raccomandata è quella di conservare in un file su disco una notevole quantità di dati, leggerli in memoria all'inizio del programma, elaborarli in memoria e riscriverli sul disco aggiornati alla fine dell'elaborazione; in tale modo si lavora molto più rapidamente che leggendo i dati da disco record per record. E' stata annunciata un'espansione ulteriore della memoria RAM del COMMODORE 128 che consentirà questo tipo di elaborazioni su una quantità ancora maggiore di dati.

Nella pagina zero della memoria, comune a tutti i banchi, si trovano i puntatori alle diverse zone della memoria del banco 1; li ripetiamo qui per comodità:
47/48 ($2F/$30) puntatore a inizio zona variabili, dopo un reset contiene 1024;
49/50 ($31/$32) puntatore a inizio zona ARRAY, dopo un reset contiene 1024;
51/52 ($33/$34) puntatore a inizio zona libera, dopo un reset contiene 1024;
53/54 ($35/$36) puntatore a inizio zona dinamica per le stringhe, gestita per indirizzi decrescenti, dopo un reset contiene 65280;
55/56 ($37/$38) puntatore alla stringa usata per ultima, dopo un reset contiene 65278
57/58 ($39/$3A) puntatore alla fine della zona dinamica per le stringhe, dopo un reset contiene 65280.

Durante lo svolgimento del programma i valori dei puntatori cambiano; in sostanza si hanno a disposizione tre zone:
- la prima per le variabili singole;
- la seconda per gli ARRAY;
- la terza per la parte variabile delle stringhe.

Mentre le prime due zone sono gestite per indirizzi crescenti, e, quando viene aggiunta una variabile singola tutta la zona degli ARRAY viene spostata verso l'alto, la zona dinamica per le stringhe è gestita per indirizzi descrescenti, anche se ogni stringa è singolarmente scritta per indirizzi crescenti. Il puntatore alla memoria libera indica la fine della zona dedicata agli ARRAY, mentre il puntatore situato nei byte 57 e 58 indica il limite superiore della zona libera. Nella figura 3.7 è schematizzato l'utilizzo della memoria del banco 1.

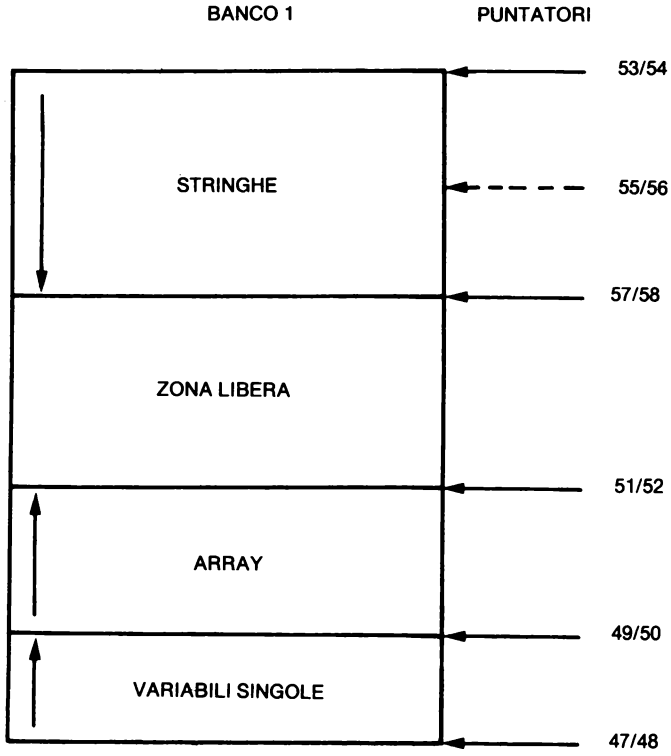


Fig. 3.7 Gestione della zona variabili in BASIC

Per vedere come sono memorizzate le variabili abbiamo preparato il programma VARINMEM, nel quale, dopo aver definito alcuni tipi di variabili ed aver assegnato loro dei valori, andiamo a leggere byte dopo byte il contenuto delle tre zone del banco 1 e lo stampiamo. Si ricordi che, anche se è consentito utilizzare nel programma nomi lunghi per le variabili, il sistema ne conserva solo i primi due caratteri e modifica i loro codici ASCII in modo da poter distinguere i diversi tipi di variabili. Inoltre si deve verificare che i nomi creati dal programmatore non contengano parole chiave, o parti di queste, che possano dar luogo ad ambiguità.
I tipi di variabili sono:
- variabili singole numeriche intere, nome seguito dal carattere %;
- variabili singole numeriche non intere, nome senza suffisso;
- variabili singole alfanumeriche (stringhe), nome seguito dal carattere $;
- variabili con indice numeriche intere, nome seguito dal carattere % e da una coppia di parentesi con gli indici;
- variabili con indice numeriche non intere, nome senza suffisso seguito da una coppia di parentesi con gli indici;
- variabili con indice alfanumeriche (stringhe con indice), nome seguito dal carattere $ e da una coppia di parentesi con gli indici.

Abbiamo verificato in memoria la struttura delle variabili che passiamo ad elencare.

VARIABILI SINGOLE NUMERICHE INTERE, nome seguito dal carattere %, sono conservati i primi due caratteri, utilizza 7 byte:
- byte 1, codice ASCII primo carattere + 128;
- byte 2, codice ASCII del secondo carattere + 128, se manca secondo carattere solo 128;
- byte 3, byte HIGH del numero intero;
- byte 4, byte LOW del numero intero;
- byte 5, 6, 7, non usati.

Come si vede le variabili singole intere sprecano spazio di memoria. Gli ultimi 3 byte possono essere utilizzati dal programmatore per memorizzare dati 'nascosti', accessibili solo conoscendone l'indirizzo e utilizzando l'istruzione POKE e la funzione PEEK.
Si ricordi che i numeri interi negativi sono conservati con il primo bit a 1 per il segno meno e in complemento a 2. I numeri positivi hanno il primo bit a zero.

VARIABILI SINGOLE NUMERICHE NON INTERE, nome senza suffisso, vengono conservati solo i primi due caratteri, utilizza 7 byte:
- byte 1, primo carattere del nome;
- byte 2, secondo carattere del nome o zero se manca;
- byte 3, esponente del numero floating point;
- byte 4, 5, 6, 7, mantissa del numero dal byte HIGH al byte LOW.

Si noti che la mantissa del numero viene memorizzata in valore e segno, cioè il primo bit di sinistra del byte HIGH a 0 significa numero positivo, a 1 numero negativo. Inoltre la mantissa viene conservata senza il primo bit significativo, che è sempre 1, in seguito alla normalizzazione; in conseguenza per ottenere il valore si deve:
- togliere il bit di sinistra e sostituirlo con 1,
- calcolare l'esponente sottraendo 128,
- applicare l'esponente al valore binario;
- calcolare il valore del numero.

Questo sistema consente di avere un bit in più di precisione nel numero.
Per esempio, la variabile A, dopo l'assegnazione A=50 appare così in memoria in 7 byte: 65 0 134 72 0 0 0,

65 0, nome,
134, esponente (caratteristica), 134-128=6, dobbiamo tener buoni 6 bit della mantissa,
72 0 0 0, mantissa, in binario 01001000, togliamo il primo bit e lo sostituiamo con un bit 1, otteniamo 11001000, prendiamo 6 bit, otteniamo 110010, trasformiamo in decimale 32+16+2=50.
Invece dopo l'assegnazione A=-50 troviamo: 65 0 134 200 0 0 0,
65 0, nome,
134, esponente, come sopra, devono essere considerati 6 bit,
200 0 0 0, mantissa, in binario 11001000, dobbiamo togliere il primo bit che dice che il numero è negativo e sostituirlo con un bit 1, quindi resta ancora 11001000, prendiamo 6 bit e otteniamo 110010, che dà 50 in decimale, tenendo conto del segno si ha -50.

VARIABILI SINGOLE ALFANUMERICHE (STRINGHE), nome seguito dal carattere $, conserva solo i primi due caratteri, utilizza 7 byte;
- byte 1: codice ASCII del primo carattere del nome;
- byte 2: codice ASCII del secondo carattere del nome +128, se manca solo 128;
- byte 3: contatore dei caratteri che compongono il corpo della stringa, massimo 255;
- byte 4: byte LOW del puntatore al corpo della stringa;
- byte 5: byte HIGH del puntatore al corpo della stringa;
- byte 6, 7: non usati.

Come si vede anche qui restano inutilizzati due byte. Al momento della creazione della stringa viene creata questa *entrata* o *testata* e la stringa viene posta nella zona dinamica, registrando nella testata la lunghezza attuale e la posizione. In seguito ogni volta che viene assegnato un nuovo valore alla stringa, anche se la lunghezza resta immutata, essa viene posizionata nuovamente nella zona dinamica e viene marcata la vecchia registrazione a significare che non è più valida. Nella zona dinamica vengono registrati tutti i caratteri del corpo della stringa seguiti da due byte, nella forma byte LOW e byte HIGH, che costituiscono il puntatore alla testata della stringa. Questi due byte, quando il corpo della stringa diventa inattuale, vengono sostituiti dai valori N 255, dove N indica il numero dei byte che lo precedono e sono inutilizzati. Questa marcatura delle zone inutilizzate consente al sistema di realizzare l'operazione di compattamento della zona dinamica quando manca spazio, operazione chiamata *garbage collection*.

VARIABILI NUMERICHE INTERE CON INDICE, valgono le stesse regole delle corrispondenti variabili singole per i due byte del nome, infatti non si può avere ambiguità dato che sono memorizzate in una zona diversa;
- byte 1, 2: nome;
- byte 3: byte LOW del numero totale dei byte occupati;
- byte 4: byte HIGH del numero totale dei byte occupati;
- byte 5: numero delle dimensioni, massimo 255;
- una coppia di byte come contatore, nella forma LOW-HIGH, dell'ultima dimensione (valore del relativo indice di definizione + 1);
- una coppia di byte come sopra per la penultima dimensione;
- coppie di byte come sopra fino alla coppia relativa alla prima dimensione;
- una coppia di byte nella forma HIGH-LOW per ogni elemento.

Come si vede qui non si ha spreco di memoria, infatti ogni elemento occupa due byte.

VARIABILI NUMERICHE NON INTERE CON INDICE, valgono le stesse regole delle

corrispondenti variabili singole per i due byte del nome, infatti non si puó avere ambiguità dato che sono memorizzate in una zona diversa;
- byte 1, 2: nome;
- byte 3: byte LOW del numero totale dei byte occupati;
- byte 4: byte HIGH del numero totale dei byte occupati;
- byte 5: numero delle dimensioni, massimo 255;
- una coppia di byte come contatore, nella forma LOW-HIGH, dell'ultima dimensione (valore del relativo indice di definizione + 1);
- una coppia di byte come sopra per la penultima dimensione;
- coppie di byte come sopra fino alla coppia relativa alla prima dimensione;
- una cinquina di byte per ogni elemento, espresso con la stessa modalità vista per le variabili singole dello stesso tipo.

VARIABILE ALFANUMERICHE CON INDICE, valgono le stesse regole delle corrispondenti variabili singole per i due byte del nome, infatti non si puó avere ambiguità dato che sono memorizzate in una zona diversa;
- byte 1, 2: nome;
- byte 3: byte LOW del numero totale dei byte occupati;
- byte 4: byte HIGH del numero totale dei byte occupati;
- byte 5: numero delle dimensioni, massimo 255;
- una coppia di byte come contatore, nella forma LOW-HIGH, dell'ultima dimensione (valore del relativo indice di definizione + 1);
- una coppia di byte come sopra per la penultima dimensione;
- coppie di byte come sopra fino alla coppia relativa alla prima dimensione;
- una terzina di byte per ogni elemento, utilizzati per contenere nel primo byte il contatore del numero di caratteri, al massimo 255, e negli altri due il puntatore al corpo della stringa nella forma LOW-HIGH.

Utilizzando tutte le notizie sopra esposte abbiamo preparato il programma VARINMEM. Esso definisce nell'ordine:
- 11 variabili singole: M, N, Y, Z, A, K, B%, I, C$, J, Y$;
- 3 variabili con indice: D(3), E%(2,3), F$(2,2,2);

e assegna valori ad esse con semplici calcoli per le numeriche, con assegnazioni di costanti per le alfanumeriche, prelevando per alcune i dati dalle linee DATA. Alcune variabili numeriche, di tipo non intero, sono utilizzate per il controllo dei cicli FOR (le variabili di tipo intero non possono essere usate come variabile di controllo nel ciclo FOR); esse sono come le altre variabili numeriche, infatti i dati relativi al limite della variabile di controllo e allo STEP sono memorizzate nell'area stack del BASIC.
Dopo la definizione delle variabili e l'assegnazione dei dati ad esse, il programma si compone di tre parti:
- nella prima vengono stampate su 11 righe i 7 byte di ognuna delle 11 variabili singole;
- nella seconda vengono stampati i dati dei 3 ARRAY, andando a leggere per ognuno il numero dei byte occupati nel terzo e quarto byte;
- nella terza viene stampata tutta la zona dinamica delle stringhe, così si possono vedere anche i buchi che si sono creati tutte le volte che si è cambiato il valore di Y$.

In ogni parte si utilizzano gli indirizzi forniti dai puntatori e si esegue l'istruzione BANK 1 per poter leggere le variabili dal loro banco.
Seguono il listato del programma e, nella figura 3.8, la parte iniziale e finale dei risultati.

```
1 REM VARINMEM
2 REM DEFINISCE ALCUNE VARIABILI
3 M=0:N=0:Y=0:Z=0
4 DIM D(3),E%(2,3),F$(2,2,2)
5 A=0:FOR K=1 TO 50:A=A+1:NEXT K
6 FOR K=0 TO 3:D(K)=K*3:NEXT K
7 B%=0:FOR I=1 TO 50 STEP 2:B%=B%+2:NEXT I
8 FOR K=0 TO 2:FOR I=0 TO 3
9 E%(K,I)=K*1+9:NEXT I,K
10 C$="FINE"
11 FOR K=0 TO 2:FOR J=0 TO 2:FOR I=0 TO 2
12 READ Y$:F$(K,J,I)=C$+Y$
13 NEXT I,J,K
14 REM PUNTATORE INIZIO VARIABILI NEL BANCO 1
15 Y=PEEK(47)+256*PEEK(48)
16 REM TRASFERISCE USCITA A STAMPANTE
17 OPEN 4,4:CMD 4
18 REM LEGGE VARIABILI SINGOLE DA BANCO 1
19 PRINT:PRINT "INIZIO VARIABILI";:M=Y
20 PRINT M;"**"
21 REM OGNI VARIABILE OCCUPA 7 BYTE
22 FOR I=0 TO 10:FOR N=0 TO 6:BANK 1:K=PEEK(M+N)
23 PRINT USING "####";K;:NEXT N:PRINT
24 M=M+7:NEXT I
25 REM STAMPA ARRAY DA BANCO 1
26 REM PUNTATORE A INIZIO ARRAY
27 Y=PEEK(49)+256*PEEK(50)
28 REM PUNTATORE A INIZIO ZONA LIBERA
29 Z=PEEK(51)+256*PEEK(52)
30 PRINT:PRINT "ARRAY";:PRINT Y;"**":K=Y
31 REM OGNI VARIABILE OCCUPA UN NUMERO
32 REM DI BYTE CHE COMPARE DOPO IL NOME
33 REM LEGGE I BYTE OCCUPATI DA BANCO 1
34 BANK 1:M=PEEK(K+2)+256*PEEK(K+3)
35 REM LEGGE UN ARRAY DA BANCO 1
36 I=0:FOR J=K TO K+M-1:BANK 1:N=PEEK(J)
37 PRINT USING "####";N;:I=I+1
38 IF I=12 THEN PRINT:I=0
39 NEXT J:K=K+M:IF K=Z THEN 41
40 PRINT:GOTO 34
41 PRINT:PRINT "FINE ARRAY: ";PEEK(Z)
42 PRINT:PRINT "CORPO STRINGHE"
43 REM PUNTATORE INIZIO ZONA DINAMICA STRINGHE
44 Y=PEEK(53)+256*PEEK(54)
45 REM PUNTATORE FINE ZONA DINAMICA STRINGHE
46 Z=PEEK(57)+256*PEEK(58)
47 REM STAMPA TRE BYTE PER RIGA
48 REM NELLA FORMA: INDIRIZZO * CONTENUTO
```

```
49 I=0:FOR K=Y TO Z:BANK 1:N=PEEK(K)
50 PRINT K;"*";:PRINT USING "####";N;
51 I=I+1:IF I=3 THEN PRINT:I=0
52 NEXT K
53 REM CHIUDE LA STAMPANTE E
54 REM RIMANDA USCITA AL VIDEO
55 PRINT#4:CLOSE 4:STOP
56 REM DATI PER STRINGHE
57 DATA A01,A02,A03,A04,A05,A06,A07,A08
58 DATA A09,A10,A11,A12,A13,A14,A15,A16
59 DATA A17,A18,A19,A20,A21,A22,A23,A24
60 DATA A25,A26,A27
```

```
INIZIO VARIABILI 1024 **
  77   0 139   0   0   0   0
  78   0 130  64   0   0   0
  89   0 139   0   0   0   0
  90   0   0   0   0   0   0
  65   0 134  72   0   0   0
  75   0   0   0   0   0   0
 194 128   0  50   0   0   0
  73   0 131  96   0   0   0
  67 128   4 250 254   0   0
  74   0 130  64   0   0   0
  89 128   3 137 253   0   0

ARRAY 1101 **
  68   0  27   0   1   0   4   0  64   0   0   0
 130  64   0   0   0 131  64   0   0   0 132  16
   0   0   0
 197 128  33   0   2   0   4   0   3   0   9   0
  10   0  11   0   9   0  10   0  11   0   9   0
  10   0  11   0   9   0  10   0  11
  70 128  92   0   3   0   3   0   3   0   3   7
 236 254   7 110 254   7 240 253   7 194 254   7
  68 254   7 198 253   7 152 254   7  26 254   7
 156 253   7 222 254   7  96 254   7 226 253   7
 180 254   7  54 254   7 184 253   7 138 254   7
  12 254   7 142 253   7 208 254   7  82 254   7
 212 253   7 166 254   7  40 254   7 170 253   7
 124 254   7 254 253   7 128 253
FINE ARRAY:  0

CORPO STRINGHE
 64896 *  70 64897 *  73 64898 *  78
 64899 *  69 64900 *  65 64901 *  50
 64902 *  55 64903 * 226 64904 *   4
```

```
64905 *  65 64906 *  50 64907 *  55
64908 *  72 64909 *   4 64910 *  70
64911 *  73 64912 *  78 64913 *  69
64914 *  65 64915 *  50 64916 *  54
64917 * 199 64918 *   4 64919 *  65
64920 *  50 64921 *  54 64922 *   3
64923 * 255 64924 *  70 64925 *  73
64926 *  78 64927 *  69 64928 *  65
64929 *  50 64930 *  53 64931 * 172
64932 *   4 64933 *  65 64934 *  50
64935 *  53 64936 *   3 64937 * 255
64938 *  70 64939 *  73 64940 *  78
64941 *  69 64942 *  65 64943 *  50
64944 *  52 64945 * 217 64946 *   4
64947 *  65 64948 *  50 64949 *  52
64950 *   3 64951 * 255 64952 *  70
64953 *  73 64954 *  78 64955 *  69
64956 *  65 64957 *  50 64958 *  51
64959 * 190 64960 *   4 64961 *  65
64962 *  50 64963 *  51 64964 *   3
64965 * 255 64966 *  70 64967 *  73
64968 *  78 64969 *  69 64970 *  65
64971 *  50 64972 *  50 64973 * 163
65223 *  48 65224 *  52 65225 * 157
65226 *   4 65227 *  65 65228 *  48
65229 *  52 65230 *   3 65231 * 255
65232 *  70 65233 *  73 65234 *  78
65235 *  69 65236 *  65 65237 *  48
65238 *  51 65239 * 202 65240 *   4
65241 *  65 65242 *  48 65243 *  51
65244 *   3 65245 * 255 65246 *  70
65247 *  73 65248 *  78 65249 *  69
65250 *  65 65251 *  48 65252 *  50
65253 * 175 65254 *   4 65255 *  65
65256 *  48 65257 *  50 65258 *   3
65259 * 255 65260 *  70 65261 *  73
65262 *  78 65263 *  69 65264 *  65
65265 *  48 65266 *  49 65267 * 148
65268 *   4 65269 *  65 65270 *  48
65271 *  49 65272 *   3 65273 * 255
65274 *  70 65275 *  73 65276 *  78
65277 *  69 65278 *  58 65279 *   4
65280 * 127
```

Fig. 3.8 Parte dei risultati del programma VARINMEM

3.4 OVERLAY DI PROGRAMMI

Il COMMODORE 128 dispone di molta memoria sia per il programma che per le variabili e per questa ragione non risulta spesso necessario ricorrere all'overlay (cioè allo scambio, un programma va a sostituire nella memoria il programma precedente che lo ha richiamato) dei programmi in memoria. Questo puó comunque essere facilmente ottenuto dal momento che quando un programma viene caricato da un altro non viene eseguito un CLR della memoria delle variabili.
Si ricordi che un programma caricato da un altro parte automaticamente.
Inoltre si puó desiderare di scrivere una procedura stendendo un programma di lancio che carica e lancia tutta una catena di altri programmi. In tale caso conviene definire tutte le variabili comuni nel programma di lancio e utilizzarle in tutti gli altri, ma ovviamente non si deve mai eseguire l'istruzione CLR.

Abbiamo preparato tre brevi programmi: LANCIO, CORTO e LUNGO, che si comportano così:
– carichiamo il programma LANCIO e diamo il comando RUN;
– LANCIO crea e stampa alcune variabili ponendo in particolare la variabile X a zero, poi carica il programma CORTO, che parte automaticamente;
– CORTO crea alcune variabili, stampa le variabili di LANCIO e le sue e, se X=0, come è la prima volta, carica il programma LUNGO, che parte automaticamente;
– LUNGO crea alcune variabili, stampa le variabili ereditate da LANCIO e da CORTO, pone X=1 e carica il programma CORTO, che parte automaticamente;
– CORTO crea alcune variabili, stampa le variabili di LANCIO e le sue, e, dato che trova X=1 si ferma.
Seguono i listati dei tre programmi e nella figura 3.9 i risultati ottenuti.

```
1 REM LANCIO
3 OPEN 4,4: CMD 4
5 A$="PROVA DI OVERLAY"
7 B$="VARIABILE DEFINITA DA LANCIO"
9 X=0:PRINT "VARIABILI DEFINITE DA LANCIO"
11 PRINT "A$=";A$:PRINT "B$=";B$
13 PRINT "X=";X
15 PRINT"CARICO PROGRAMMA CORTO"
17 PRINT#4:CLOSE 4
19 DLOAD"CORTO"
21 END
```

```
1 REM CORTO
3 REM SI DEVONO MANTENERE LE VARIABILI
5 C$="SCRIVI CC$:"
7 PRINT C$;:INPUT CC$
9 OPEN 4,4:CMD 4
11 PRINT "RUN CORTO"
13 A=123:B=678
15 PRINT "VARIABILI EREDITATE DA LANCIO"
```

```
17 PRINT "A$=";A$:PRINT "B$=";B$:PRINT"X=";X
19 PRINT "VARIABILI NUOVE DI CORTO"
21 PRINT "A=";A,"B=";B
23 PRINT "C$=";C$
25 PRINT "CC$=";CC$
27 PRINT "CARICO LUNGO SE X=0"
29 PRINT#4:CLOSE 4
31 IF X=0 THEN DLOAD "LUNGO"
33 PRINT "FINITA PROVA OVERLAY"
35 END
```

```
1 REM LUNGO
3 REM DEVONO MANTENERSI TUTTE LE VARIABILI
5 REM QUELLE DEFINITE DA LANCIO E DA CORTO
7 OPEN4,4:CMD4
9 PRINT"RUN LUNGO"
11 D=998
13 E=790
15 F$="LUNGO"
17 PRINT "VARIABILI EREDITATE DA LANCIO"
19 PRINT "A$=";A$:PRINT "B$=";B$
21 PRINT "X=";X
23 PRINT "VARIABILI EREDITATE DA CORTO"
25 PRINT"A=";A,"B=";B
27 PRINT "C$=";C$
29 PRINT"CC$=";CC$
31 PRINT "VARIABILI NUOVE DI LUNGO"
33 PRINT"D=";D,"E=";E
35 PRINT"F$=";F$
37 X=1:PRINT "MODIFICA DI X: X=";X
39 PRINT#4:CLOSE4
41 DLOAD "CORTO"
43 END
```

```
VARIABILI DEFINITE DA LANCIO
A$=PROVA DI OVERLAY
B$=VARIABILE DEFINITA DA LANCIO
X= 0
CARICO PROGRAMMA CORTO


RUN CORTO
VARIABILI EREDITATE DA LANCIO
A$=PROVA DI OVERLAY
B$=VARIABILE DEFINITA DA LANCIO
X= 0
VARIABILI NUOVE DI CORTO
A= 123            B= 678
C$=SCRIVI CC$:
CC$=STO PROVANDO OVERLAY
CARICO LUNGO SE X=0


RUN LUNGO
VARIABILI EREDITATE DA LANCIO
A$=PROVA DI OVERLAY
B$=VARIABILE DEFINITA DA LANCIO
X= 0
VARIABILI EREDITATE DA CORTO
A= 123            B= 678
C$=SCRIVI CC$:
CC$=STO PROVANDO OVERLAY
VARIABILI NUOVE DI LUNGO
D= 998            E= 790
F$=LUNGO
MODIFICA DI X: X= 1


RUN CORTO
VARIABILI EREDITATE DA LANCIO
A$=PROVA DI OVERLAY
B$=VARIABILE DEFINITA DA LANCIO
X= 1
VARIABILI NUOVE DI CORTO
A= 123            B= 678
C$=SCRIVI CC$:
CC$=SECONDA VOLTA DI CORTO
CARICO LUNGO SE X=0
```

Fig. 3.9 Risultati dei programmi LANCIO, CORTO e LUNGO

CAPITOLO 4

ASSEMBLER E LINGUAGGIO MACCHINA

4.1 PROGRAMMAZIONE IN LINGUAGGIO MACCHINA

Per eseguire un programma in linguaggio macchina si devono scrivere in memoria in una sequenza di byte i numeri esadecimali (o i corrispondenti decimali) che rappresentano le istruzioni riconosciute dal microprocessore centrale, codici operativi e relativi operandi, e nel registro PC (Program Counter) l'indirizzo del primo byte del programma. Il COMMODORE 128, in modo 128, ci consente di scrivere in memoria un programma già in linguaggio macchina utilizzando i comandi del MONITOR, e di eseguirlo.

In realtà scrivere un programma direttamente in linguaggio macchina risulta piuttosto noioso, a meno che non si tratti di poche istruzioni, e quindi si ricorre di solito al linguaggio simbolico ASSEMBLER, il quale mette a disposizione codici operativi e operandi simbolici. Per poter programmare in ASSEMBLER è necessario disporre di un programma assemblatore che traduce il programma sorgente, scritto in codice simbolico, nel programma oggetto in linguaggio macchina.

Il MONITOR del COMMODORE 128 ci mette a disposizione un *mini assemblatore*, possiamo cioè scrivere un programma utilizzando i codici mnemonici delle istruzioni, ma non gli operandi simbolici e le varie *istruzioni direttive* che in generale si utilizzano nel linguaggio ASSEMBLER per riservare aree di memoria o altro.
Comunque quello che ci offre il MONITOR è già un bel vantaggio, possiamo infatti scrivere gli operandi sia utilizzando numeri esadecimali che decimali (ponendo come primo carattere il +), ma non dobbiamo preoccuparci di scrivere prima la parte LOW e poi la parte HIGH degli indirizzi.

Per scrivere un programma in questo ASSEMBLER ridotto dobbiamo come sempre procedere nell'analisi del problema, definire le variabili da utilizzare, alle quali possiamo per comodità assegnare nomi simbolici nella fase di studio.
Dopo aver definito lo schema del programma, tenendo presente che non verrà codificato in BASIC, dobbiamo decidere un indirizzo di memoria dal quale far iniziare il programma e gli indirizzi di memoria per le variabili. Puó essere raccomandabile tenere tutte le variabili in un blocco unico di byte consecutivi, che puó precedere o seguire il programma. A nostro avviso la cosa più semplice, dopo aver definito la zona di memoria da utilizzare, è fare un calcolo approssimato per eccesso dei byte necessari per le variabili e sistemarle all'inizio della zona di memoria; in tale modo potranno restare alcuni byte inutilizzati tra l'ultima variabile e il programma, ma non si rischia di dover modificare gli indirizzi utilizzati nel programma prima di terminarlo.
All'interno del programma siamo costretti ad utilizzare quasi sempre indirizzi assoluti, per esempio nelle istruzioni di salto che iniziano con B e che ammettono il modo di indirizzamento relativo, noi dobbiamo scrivere l'indirizzo vero dove vogliamo saltare se è verificata la condizione, il codice prodotto contiene invece l'indiriz-

zo relativo, cioè lo spiazzamento necessario per raggiungere quell'indirizzo. Per esempio, supponiamo di essere arrivati nella stesura del programma all'indirizzo $4000 e di dover utilizzare un'istruzione di salto condizionato tipo BEQ, che produce un salto se l'operazione precedente ha dato un risultato nullo, per andare 32 byte ($20) più avanti, noi scriviamo:

```
...
A 04000 BEQ 4020
```

e vediamo comparire sul video:

```
...
A 04000 F0 1E BEQ $4020
A 04002
```

dove F0 è il codice esadecimale di BEQ e 1E (30 decimale) rappresenta lo spiazzamento, cioè il numero da aggiungere al PC per raggiungere la posizione voluta.
Programmando con un linguaggio ASSEMBLER completo avremmo in questo caso utilizzato un riferimento simbolico, tipo I5, o altro, e avremmo riportato tale parola (label) come label dell'istruzione destinazione del salto; il programma assemblatore avrebbe calcolato lo spiazzamento necessario.
Inoltre molti indirizzi sono assoluti, come destinazioni di salti JMP o indirizzi di dati; in conseguenza si raccomanda di scrivere il programma dopo aver deciso dove posizionare in memoria esso e le sue variabili.
Durante la stesura del codice si lasciano momentaneamente in sospeso gli indirizzi che non sono ancora noti (essenzialmente destinazioni di salti condizionati e non), e si aggiungono mano mano che si procede.

In generale, dato che il BASIC 7.0 del COMMODORE 128 è molto potente e ricco di istruzioni per la grafica, il movimento degli *sprite* e il suono, non si dovrà ricorrere spesso alla programmazione in linguaggio macchina. A volte si tratterà di scrivere piccole routine in linguaggio macchina da utilizzare nei programmi in BASIC e si dovrà scegliere in quale zona di memoria posizionarle.

Le routine in linguaggio macchina utilizzate da un programma BASIC devono terminare con l'istruzione RTS, che restituisce il controllo al BASIC.

Studiando la mappa della memoria del banco 0 si trovano diverse zone utilizzabili, come il buffer della cassetta, se non si usa la cassetta, il buffer della porta RS-232, se non si usa tale porta. Un'altra cosa che si puó fare è riservare spazio per i programmi in linguaggio macchina o prima della zona dedicata al programma in BASIC o dopo. Per riservare spazio prima bisogna modificare il contenuto dei due byte 43 e 44 che costituiscono il puntatore all'inizio del programma BASIC, per riservare spazio dopo, invece, bisogna modificare il contenuto dei due byte 4626 e 4627 che costituiscono il puntatore alla fine della memoria disponibile per il programma BASIC nel banco 0. Tali modifiche dei puntatori possono essere eseguite dal programma stesso nelle sue prime linee.

Vediamo ora come il programma BASIC puó caricare il codice macchina. Il codice macchina viene prodotto utilizzando il MONITOR; esso puó essere visualizzato sul video e anche stampato su carta. Se prima di attivare il MONITOR si trasferisce l'uscita dal video alla stampante con: OPEN 4,4:CMD 4, tutto l'output va alla stampante e puó essere conservato. In conseguenza è possibile trascrivere in linee DATA del programma BASIC il valore dei byte che contengono il linguaggio macchina in codice esadecimale, considerandoli come stringhe (non sono accettati

i caratteri esadecimali come numeri in BASIC), e puó essere incorporata nel programma una routine che legge tali stringhe, le trasforma in numeri decimali e memorizza i numeri nella zona destinata al linguaggio macchina; la cosa risulta estremamente semplice utilizzando la funzione DEC. Se si vuole si possono trasformare i numeri esadecimali in decimali (uno per volta) utilizzando il MONITOR stesso, oppure si puó operare manualmente il cambiamento di codice e porre nelle linee DATA già numeri decimali.
Il metodo più efficiente è utilizzare ancora il MONITOR per memorizzare il codice macchina in un file binario o su cassetta o su disco. Questo metodo consente di far iniziare il programma BASIC con le eventuali istruzioni necessarie per riservare lo spazio per il codice macchina, seguite dall'istruzione BLOAD che carica il file binario in memoria nella posizione voluta.

Per eseguire i programmi in linguaggio macchina il BASIC mette a disposizione l'istruzione SYS e la funzione USR.
Se è necessaria una comunicazioni di dati o risultati tra il BASIC e le routine in linguaggio macchina si possono scegliere diversi metodi, che descriviamo nel seguito.

– Servirsi di locazioni fisse di memoria, raggiungibili dal BASIC con l'istruzione POKE e la funzione PEEK, e dal linguaggio macchina tramite l'indirizzo.

– Utilizzare i parametri opzionali dell'istruzione SYS per caricare valori rispettivamente nei registri A, X, Y e P. Infatti il formato completo dell'istruzione è:

SYS ind,na,nx,ny,np

dove i parametri hanno il seguente significato:
ind, indirizzo di inizio della routine in linguaggio macchina;
na, numero (massimo 255) da caricare nell'accumulatore A;
nx, numero (massimo 255) da caricare nel registro X;
ny, numero (massimo 255) da caricare nel registro Y;
np, numero (massimo 255) da caricare nel registro di stato P (registro dei flag), facendo attenzione a scegliere un valore che non provochi segnalazione di errore.

– Utilizzare la funzione RREG, che legge ordinatamente il contenuto dei registri A, X, Y e P, per trasferire il valore di tali registri in variabili del programma BASIC. L'istruzione si scrive:

RREG N1, N2, N3, N4

dove le 4 variabili ricevono ordinatamente i contenuti dei 4 registri sopra citati. Se alla RREG si fa seguire:

PRINT N1, N2, N3, N4

viene visualizzato il contenuto dei 4 registri. Se si utilizzano meno variabili si ottengono solo i contenuti dei primi registri.

– Utilizzare la funzione USR ponendo tra parentesi una espressione valida; tale espressione viene calcolata dal sistema e posta nell'accumulatore floating point FAC1. Il programma in linguaggio macchina, il cui indirizzo di inizio deve essere stato memorizzato nei due byte 4633 ($1219) e 4634 ($121A) prima di utilizzare

la funzione USR, puó lavorare sul dato che trova in FAC1. Alla fine il valore di FAC1 viene ritornato al programma BASIC tramite la variabile di chiamata; per esempio scrivendo:

100 A=USR(X+9)

si ottiene che:
– il numero X+9 viene memorizzato in FAC1,
– il programma salta alla routine il cui indirizzo si trova nei byte 4633 e 4634 nella forma byte LOW, byte HIGH, e la esegue,
– quando il controllo torna al programma BASIC per effetto dell'istruzione RTS finale della routine, la variabile BASIC A contiene il valore di FAC1. Tale valore puó essere quello iniziale o un altro a seconda delle operazioni eseguite dalla routine attivata da USR.

4.2 LE ISTRUZIONI DEL MICROPROCESSORE 8502

Il campo operando, presente nelle istruzioni, identifica il dato su cui l'istruzione deve operare. L'identificazione puó avvenire in diversi modi, chiamati *modi di indirizzamento*, che elenchiamo.

– IMPLICITO, le informazioni sull'operando sono contenute implicitamente nel codice operativo, che fa riferimento a un registro del processore; istruzioni a un solo byte.

– IMMEDIATO, l'operando è una costante scritta nel byte successivo a quello del codice operativo; istruzione a due byte.

– ASSOLUTO, l'operando è un dato contenuto in una locazione di memoria il cui indirizzo viene scritto nei due byte successivi a quello del codice operativo; istruzioni a tre byte. Si veda la figura 4.1.

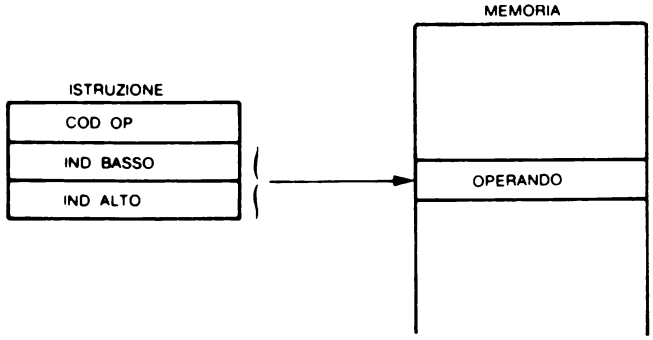


Fig. 4.1 Indirizzamento ASSOLUTO

– IN PAGINA ZERO, caso particolare di indirizzamento assoluto, in quanto l'indirizzo occupa un solo byte; istruzioni a due byte.
Si veda la figura 4.2.

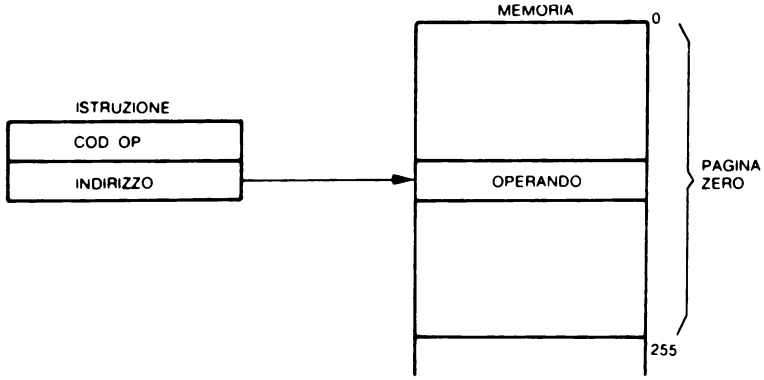


Fig. 4.2 Indirizzamento IN PAGINA ZERO

– RELATIVO, il byte successivo a quello del codice operativo contiene un numero che indica la distanza (spiazzamento, da -128 a +127)) dalla posizione attuale; l'indirizzo si ottiene sommando lo spiazzamento al registro PC. Istruzioni a due byte. Si veda la figura 4.3.

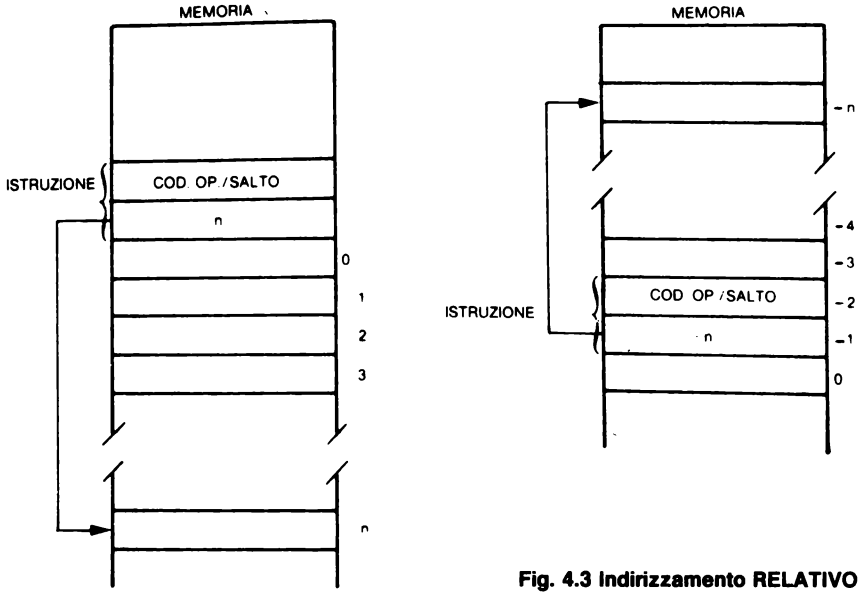


Fig. 4.3 Indirizzamento RELATIVO

– INDIRETTO, il byte successivo a quello del codice operativo contiene un indirizzo della pagina zero; a tale indirizzo e nel byte successivo si trova l'indirizzo dell'operando. Istruzioni a due byte. Si veda la figura 4.4.

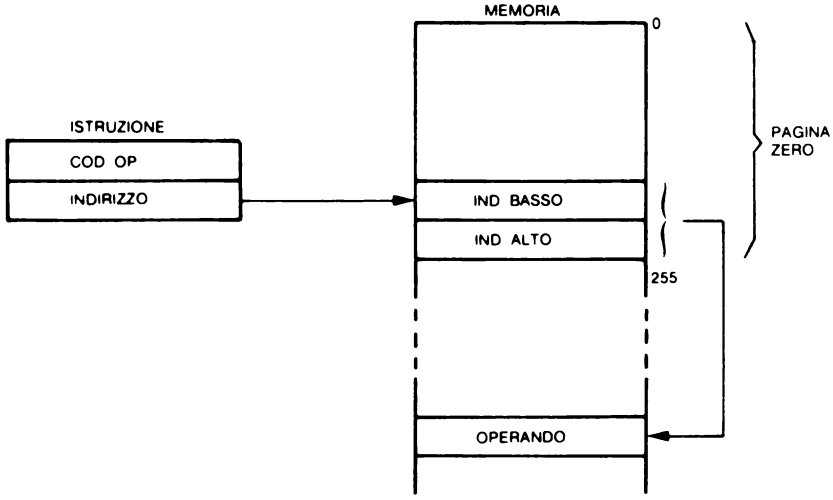


Fig. 4.4 Indirizzamento INDIRETTO

– ASSOLUTO INDICIZZATO TRAMITE REGISTRO X, i due byte successivi a quello del codice operativo contengono un numero che sommato al contenuto del registro X fornisce l'indirizzo dell'operando. Istruzioni a tre byte. Si veda la figura 4.5.

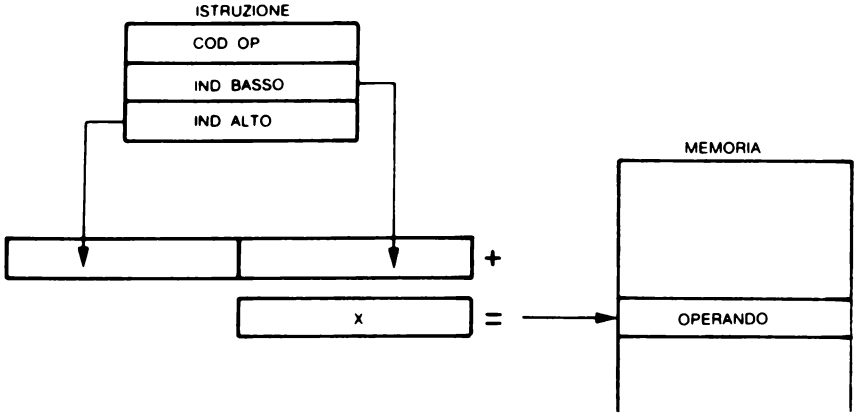


Fig. 4.5 Indirizzamento ASSOLUTO INDICIZZATO TRAMITE REGISTRO X

– ASSOLUTO INDICIZZATO TRAMITE REGISTRO Y, come il precedente, ma utilizzando il registro Y.

– INDICIZZATO IN PAGINA ZERO TRAMITE IL REGISTRO X, analogo al precedente, solo che il valore da sommare al registro X occupa un solo byte; istruzioni a due byte. Si veda la figura 4.6.

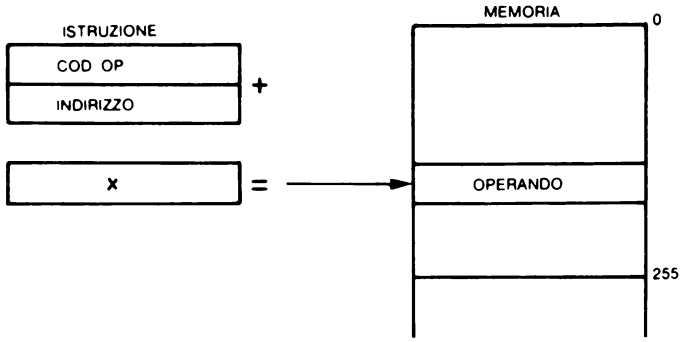


Fig. 4.6 Indirizzamento INDICIZZATO IN PAGINA ZERO TRAMITE IL REGISTRO X

– INDICIZZATO IN PAGINA ZERO TRAMITE IL REGISTRO Y, come il precedente, ma utilizzando il registro Y.

– INDICIZZATO INDIRETTO, il byte successivo a quello del codice operativo contiene un valore che sommato al contenuto del registro X dà un indirizzo della pagina zero. Il byte indirizzato e il successivo contengono l'indirizzo dell'operando. Istruzioni a due byte. Si veda la figura 4.7.

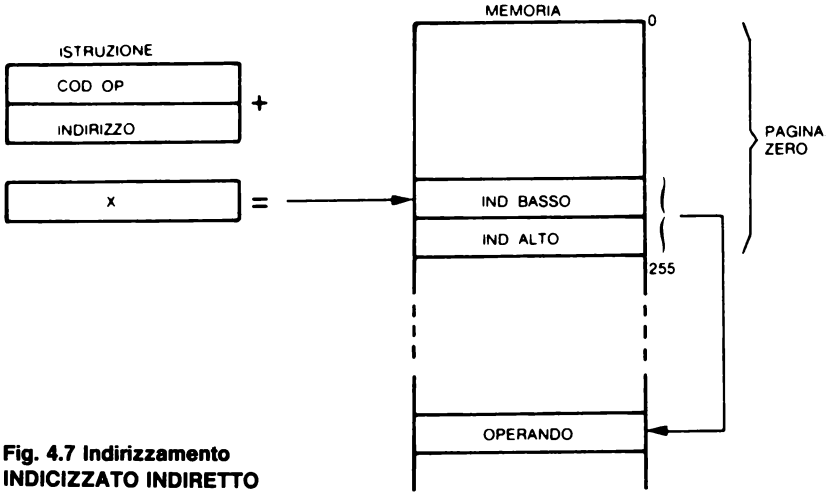


Fig. 4.7 Indirizzamento INDICIZZATO INDIRETTO

– INDIRETTO INDICIZZATO, il byte successivo a quello del codice operativo contiene un indirizzo della pagina zero. Il byte indirizzato e il successivo in pagina zero contengono un indirizzo al quale va aggiunto il contenuto del registro Y per ottenere l'indirizzo dell'operando. Istruzioni a due byte. Si veda la figura 4.8.

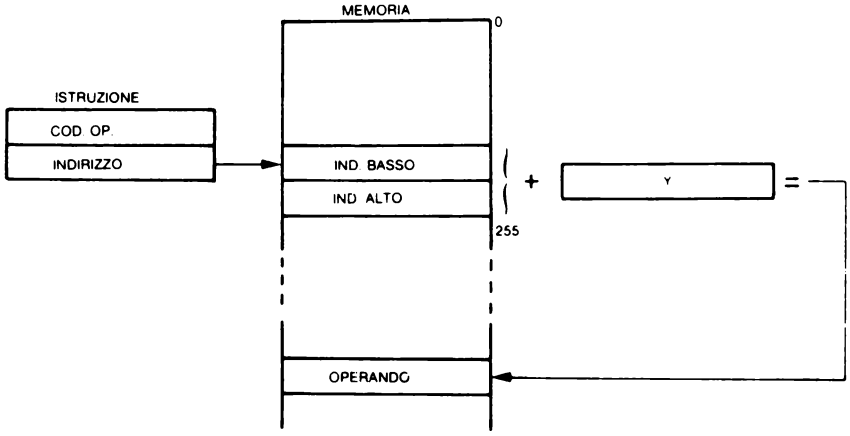


Fig. 4.8 Indirizzamento INDIRETTO INDICIZZATO

I registri a disposizione del programmatore sono schematizzati nella figura 4.9; essi sono:
– registro indice X, di 8 bit;
– registro indice Y, di 8 bit;
– registro S puntatore all'area stack, di 8 bit;
– registro A accumulatore, di 8 bit;
– registro PCL, LOW byte del PC (Program Counter), contatore del programma, di 8 bit;
– registro PCH, HIGH byte del PC (Program Counter), contatore del programma, di 8 bit;
– registro P (Processor Status Register), registro dei FLAG, di 8 bit.

Il registro PC, formato dai due registri PCL e PCH, è di 16 bit e contiene l'indirizzo della prossima istruzione da eseguire. Esso viene automaticamente incrementato durante l'esecuzione di una istruzione.

I registri X e Y possono essere utilizzati per la formazione degli indirizzi degli operandi ed esistono delle istruzioni apposite che operano su di essi.

Il registro A è uno degli operandi delle istruzioni aritmetiche e logiche. Sui registri della memoria si puó operare in modo aritmetico solo incrementandone o decrementandone il contenuto di 1, oppure operando divisioni o moltiplicazioni per 2 per mezzo delle operazioni di scorrimento.

Il registro S viene utilizzato per gestire una zona particolare della memoria chiamata

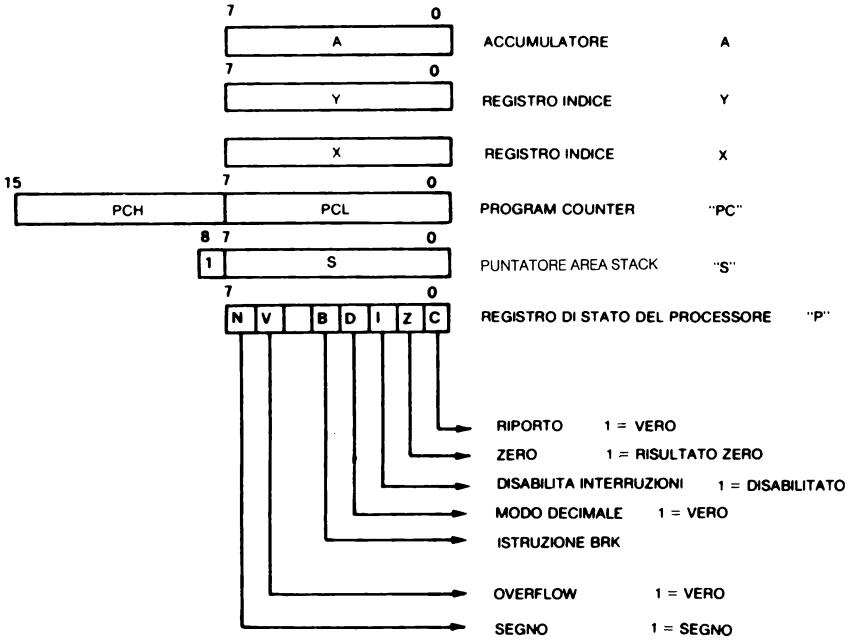


Fig. 4.9 Registri a disposizione del programmatore

STACK (catasta, pila); essa viene gestita come una pila con la tecnica LIFO (Last In First Out) e il registro S punta sempre alla prima posizione libera dell'area. Il registro S è a 8 bit e quindi contiene sempre la parte LOW dell'indirizzo dell'area stack; nel nostro caso la parte HIGH vale sempre $01. Nella figura 4.10 è schematizzata l'area stack con il registro S.

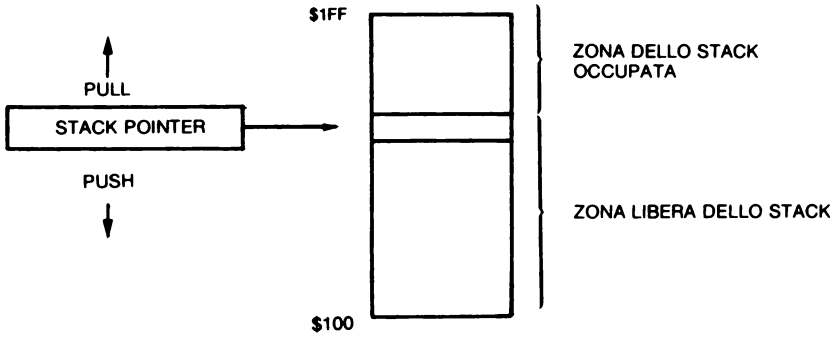


Fig. 4.10 Stack e stack pointer

Questa zona viene utilizzata per memorizzare dati o indirizzi in modo temporaneo. L'operazione di inserimento di un byte nell'area stack si chiama PUSH; il byte viene memorizzato all'indirizzo presente nel registro S e il registro S viene decrementato per puntare al prossimo byte libero. L'operazione di prelevamento dall'area stack si chiama PULL; il registro S viene incrementato e poi viene prelevato il byte dall'indirizzo indicato da S, che così punta al primo byte libero dell'area.

Il registro P ha una funzione molto importante, infatti i sui bit sono dei FLAG che segnalano fatti accaduti durante l'esecuzione delle altre istruzioni e possono essere interrogati per prendere delle decisioni sul proseguimento del programma, cioè consentono di operare delle scelte.
I bit 7, 6, 1 e 0 cambiano valore durante l'esecuzione di alcune operazioni:
– il bit 7, chiamato N, è a 1 se il risultato dell'operazione è negativo, a 0 altrimenti;
– il bit 6, chiamato V, indica se se si è avuto un traboccamento (overflow) durante l'operazione, cioè il risultato è troppo grande o troppo piccolo;
– il bit 1, chiamato Z, è a 1 se il risultato dell'operazione è zero;
– il bit 0, chiamato C, puó segnalare diversi fatti a seconda delle operazioni e indica un riporto (carry).
Il bit 3, chiamato D, viene gestito dal programmatore per attivare l'esecuzione di operazioni in modo decimale (in codice BCD).
Il bit 2, chiamato I, è gestito dal programmatore e disabilita le interruzioni se posto a 1. Alcune istruzioni lo modificano.
Il bit 4, chiamato B, corrisponde a un diverso comportamento della routine di interrupt e interessa l'istruzione BRK.

Le istruzioni possono essere ripartite nei seguenti gruppi:
– 1. trasferimento dati,
– 2. operazioni nell'area stack,
– 3. aritmetico-logiche,
– 4. rotazione o scorrimento,
– 5. confronto,
– 6. salto,
– 7. chiamata e ritorno da sottoprogramma,
– 8. controllo.

.1. TRASFERIMENTO DATI

LDA (LoaD Accumulator): carica nell'accumulatore il dato specificato dall'operando.

STA (STore Accumulator): copia all'indirizzo specificato dall'operando il contenuto dell'accumulatore.

LDX (LoaD X): carica nel registro X il dato specificato dall'operando.

STX (STore X): copia all'indirizzo specificato il contenuto del registro X.

LDY (LoaD Y): carica nel registro Y il dato specificato dall'operando.

STY (STore Y): copia all'indirizzo specificato il contenuto del registro Y.

TAX (Transfer Accumulator into X): copia il contenuto di A in X.

TXA (Transfer X into Accumulator): copia il contenuto di X in A.

TAY (Transfer Accumulator into Y): copia il contenuto di A in Y.

TYA (Transfer Y into Accumulator): copia il contenuto di Y in A.

TSX (Transfer S into X): copia il contenuto del registro S in X.

TXS (Transfer X into S): copia il contenuto del registro X in S.

.2. OPERAZIONI NELL'AREA STACK

PHA (PusH Accumulator): il contenuto dell'accumulatore viene inserito nell'area stack all'indirizzo contenutenuto in S, poi S viene decrementato di 1. Si veda la figura 4.11.

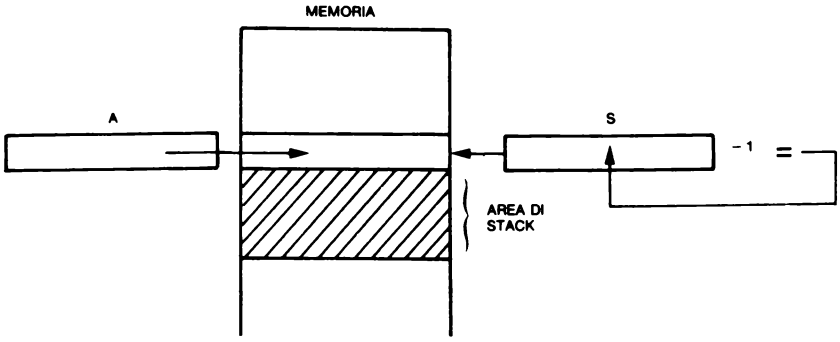


Fig. 4.11 Istruzione PHA

PLA (PulL Accumulator): il contenuto del byte puntato da S viene copiato nell'accumulatore, poi S viene incrementato di 1. Si veda la figura 4.12.

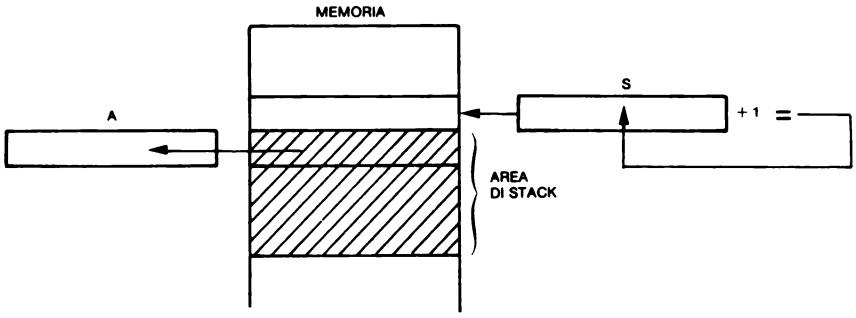


Fig. 4.12 Istruzione PLA

PHP (PusH P): il contenuto del registro di stato P viene scritto nel byte puntato da S, poi S viene decrementato di 1.

PLP (PulL P): il contenuto del byte puntato da S viene copiato nel registro di stato P, poi S viene incrementato di 1.

.3. ARITMETICO-LOGICHE

ADC (ADd with Carry): somma il contenuto dell'accumulatore all'operando, aggiunge il bit di carry e pone il risultato in A.

SBC (SuBtract with Carry): somma all'operando il complemento del carry e sottrae il risultato da A. Si veda la figura 4.13.

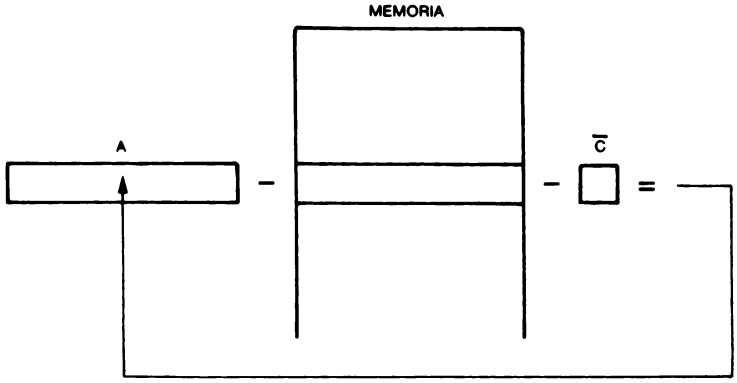


Fig. 4.13 Istruzione SBC

INC (INCrement): incrementa di 1 l'operando.

DEC (DECrement): decrementa di 1 l'operando.

INX (INcrement X): incrementa di 1 il registro X.

DEX (DEcrement X): decrementa di 1 il registro X.

INY (INcrement Y): incrementa di 1 il registro Y.

DEY (DEcrement Y): decrementa di 1 il registro Y.

AND: esegue l'operazione AND bit a bit del contenuto dell'accumulatore con l'operando e pone il risultato in A.
(0 AND 0 = 0, 0 AND 1 = 0, 1 AND 0 = 0, 1 AND 1 = 1)

ORA: esegue l'operazione OR bit a bit del contenuto dell'accumulatore con l'operando e pone il risultato in A.
(0 OR 0 = 0, 0 OR 1 = 1, 1 OR 0 = 1, 1 OR 1 = 1)

EOR: esegue l'operazione XOR (OR esclusivo) dell'operando con il contenuto dell'accumulatore e pone il risultato in A.
(0 XOR 0 = 0, 0 XOR 1 = 1, 1 XOR 0 = 1, 1 XOR 1 = 0)

.4. ROTAZIONE O SCORRIMENTO

ASL (Arithmetic Shift Left): sposta a sinistra di un bit il contenuto dell'operando, tale bit viene copiato nel carry, e inserisce un bit 0 a destra (corrisponde a moltiplicare per 2).

LSR (Logical Shift Right): sposta a destra di un bit, copia il bit 0 nel carry e inserisce un bit 0 a sinistra (corrisponde a dividere per 2).

ROL (ROtate Left): il contenuto dell'operando forma un anello con il carry, il tutto viene spostato verso sinistra di un bit, per cui il carry va nel bit 0 e il bit 7 va nel carry.

ROR (ROtate Right): come sopra ma spostando verso destra, per cui il carry va nel bit 7 e il bit 0 va nel carry.

.5. CONFRONTO

CMP (CoMPare with accumulator): confronta senza modifiche l'operando con l'accumulatore, equivale a una sottrazione ideale A - operando; vengono modificati i flag N, Z e C.

CPX (ComPare with X): come la precedente con X al posto di A.

CPY (ComPare with Y): come la precedente con Y al posto di X.

BIT (test BIT in memory): esegue l'operazione AND tra l'operando e il contenuto di A senza modificarli. Il flag Z va a 1 se il risultato è 0, cioè se sono uguali; i bit 6 e 7 dell'operando sono copiati nei flag V e N. Si veda la figura 4.14.

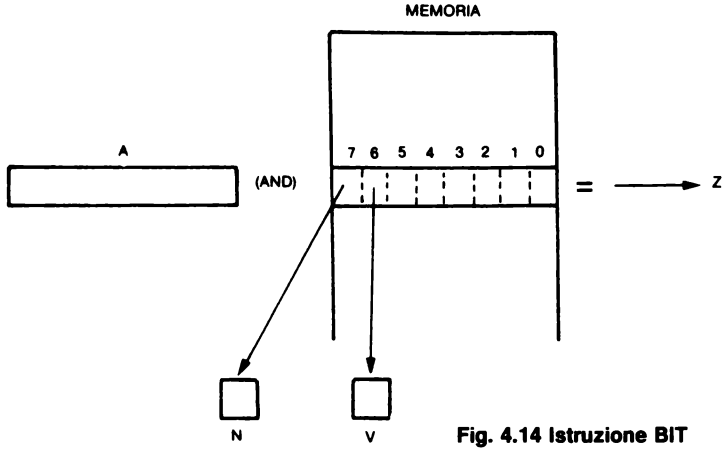


Fig. 4.14 Istruzione BIT

.6. SALTO

Tutte le operazioni il cui nome mnemonico inizia con B usano il modo di indirizzamento relativo; in conseguenza sono possibili salti all'indietro o in avanti in base allo spiazzamento, 128 byte all'indietro e 127 in avanti al massimo (il contenuto di un byte è considerato negativo se inizia con 1, complemento a 2).

BMI (Branch on Minus): trasferisce il controllo all'indirizzo specificato se il flag N si trova a 1 (risultato negativo), altrimenti prosegue. Si veda la figura 4.15.

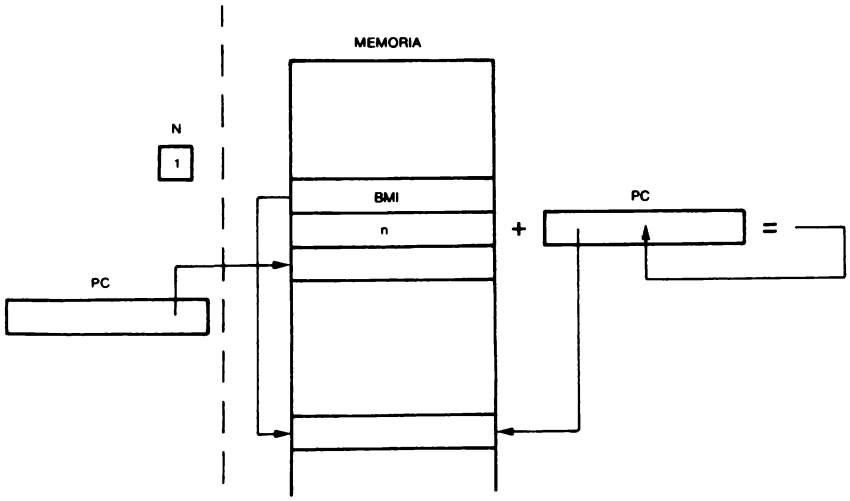


Fig. 4.15 Istruzione BMI

BPL (Branch on PLus): come la precedente se N=0, cioè risultato positivo o nullo.

BCS (Branch on Carry Set): come la precedente se C=1.

BCC (Branch on Carry Clear): come la precedente se C=0.

BEQ (Branch if EQual zero): come la precedente se Z=1, cioè se l'ultima istruzione che lo ha modificato ha dato risultato zero.

BNE (Branch on Not Equal to zero): come la precedente se Z=0, cioè se l'ultima istruzione che lo ha modificato ha dato risultato diverso da zero.

BVC (Branch on Overflow Clear): come la precedente se V=0, cioè se l'ultima istruzione che lo ha modificato non ha dato riporto dal bit 6 del risultato.

BVS (Branch on Overflow Set): come la precedente se V=1.

JMP (JuMP): viene caricato nel PC l'indirizzo specificato e quindi il programma prosegue da quell'indirizzo.

.7. CHIAMATA E RITORNO DA SOTTOPROGRAMMA

JSR (Jump to SubRoutine): viene memorizzato nell'area stack il contenuto del PC, poi viene caricato nel PC l'indirizzo specificato, cioè il programma prosegue da quell'indirizzo. E' l'istruzione di chiamata a sottoprogramma. Si veda la figura 4.16.

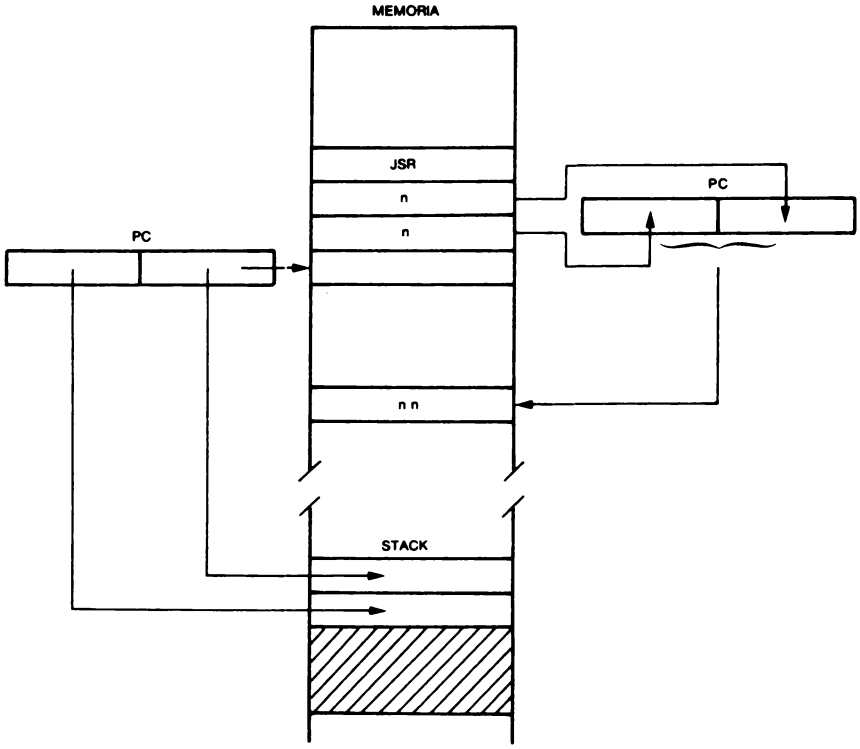


Fig. 4.16 Istruzione JSR

RTS (ReTurn from Subroutine): copia i due byte puntati dal registro S (nell'area stack) nel PC, lo incrementa e aggiorna il registro S; quindi il programma prosegue dall'indirizzo successivo a quello dell'istruzione JSR. Si veda la figura 4.17.

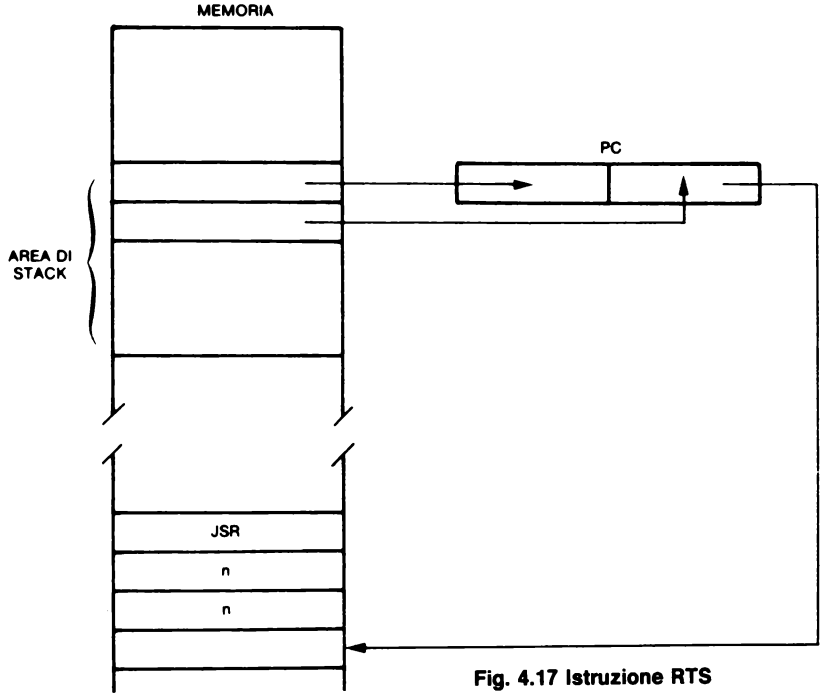


Fig. 4.17 Istruzione RTS

RTI (ReTurn from Interrupt): questa deve essere l'ultima istruzione di ogni routine di servizio di interrupt. Essa preleva i tre byte affioranti nell'area stack e li memorizza rispettivamente nel registro di stato P e nel PC e aggiorna il registro S. Si veda la figura 4.18.

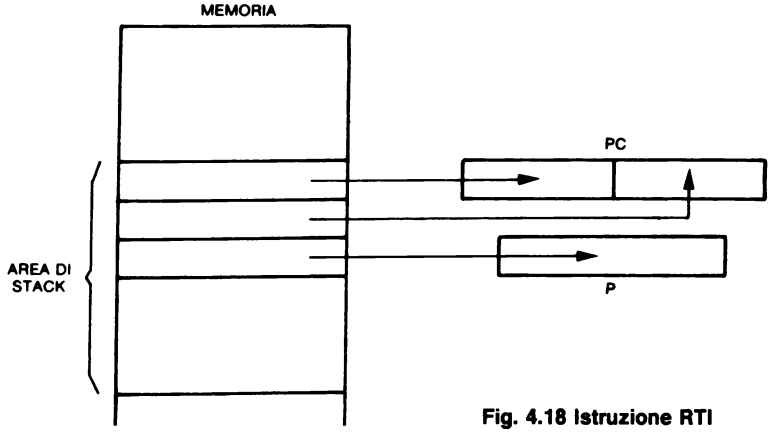


Fig. 4.18 Istruzione RTI

.8. CONTROLLO

CLC (CLear Carry): azzera il flag C.

SEC (SEt Carry): pone a 1 il flag C.

CLD (CLear Decimal): azzera il flag D; se questo flag è a zero le operazioni aritmetiche vengono eseguite su numeri binari.

SED (SEt Decimal): pone a 1 il flag D; le operazioni aritmetiche successive vengono eseguite su numeri espressi in BCD.

CLI (CLear Interrupt): azzera il flag I di disabilitazione degli interrupt; con tale valore di I gli interrupt risultano abilitati.

SEI (SEt Interrupt): pone a 1 il flag I; in tale modo gli interrupt risultano disabilitati.

CLV (CLear oVerflow): azzera il flag V di overflow; non esiste l'operazione contraria dato che V viene posto a 1 per effetto di alcune operazioni.

NOP (No OPeration): non viene eseguita alcuna operazione salvo l'aggiornamento del registro PC.

BRK (BReaK): pone a 1 il bit B e memorizza nell'area stack il contenuto del registro di stato P e del registro PC e poi esegue un salto indiretto alla locazione $0316. Viene cioè eseguita la routine di interrupt per richiesta del programma. Il bit B a 1 provoca in caso di interrupt via hardware il salto indiretto alla locazione $0314.

Nelle pagine che seguono sono elencate in ordine alfabetico del codice mnemonico le istruzioni. Per ogni istruzione viene presentata una tabella nella quale sono riportati:
- il codice mnemonico,
- uno schema di comportamento,
- il comportamento dei flag,
- i modi di indirizzamento,
- lo schema di scrittura dell'istruzione,
- il codice esadecimale,
- il numero dei byte occupati,
- il numero dei cicli macchina impiegati.

Le lettere distintive dei flag sono:
N, segno - Z, zero - C, carry - I, disabilitazione interrupt-D, aritmetica decimale - V, overflow.
Non abbiamo segnato il flag B che interessa solo l'istruzione BRK.

Nelle tabelle sono usate le seguenti convenzioni:

A	accumulatore	X,Y	registri indice
M	memoria	P	registro di stato (flag)
S	puntatore stack	m	flag modificato
/	flag non modificato	+	addizione
∧	operazione AND	–	sottrazione

∀	operazione XOR	↑	PULL da area stack
↓	PUSH in area stack	→	trasferimento
←	trasferimento	∨	operazione OR
PC	Program Counter	PCH	Program Counter HIGH
PCL	Program Counter LOW	op	operando
#	indirizz. immediato	(..)	ind. indiretto

* il numero dei cicli macchina aumenta di 1 se si esce dalla pagina corrente

** il numero dei cicli macchina aumenta di 1 se si esce dalla pagina corrente e di 1 se ha luogo il salto.

Tabella 4.1 Istruzioni del processore 8502

ADC
A+M+C → A,C

N	Z	C	I	D	V
m	m	m	/	/	m

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	ADC #op	69	2	2
pag zero	ADC op	65	2	3
pag zero,X	ADC op,X	75	2	4
ass	ADC op	6D	3	4
ass,X	ADC op,X	7D	3	4*
ass,Y	ADC op,Y	79	3	4*
(ind,X)	ADC (op,X)	61	2	6
(ind),Y	ADC (op),Y	71	2	5*

AND
A ∧ M → A

N	Z	C	I	D	V
m	m	m	/	/	m

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	AND #op	29	2	2
pag zero	AND op	25	2	3
pag zero,X	AND op,X	35	2	4
ass	AND op	2D	3	4
ass,X	AND op,X	3D	3	4*
ass,Y	AND op,Y	39	3	4*
(ind,X)	AND (op,X)	21	2	6
(ind),Y	AND (op),Y	31	2	5

ASL

N	Z	C	I	D	V
m	m	m	/	/	m

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
acc	ASL A	0A	1	2
pag zero	ASL op	06	2	5
pag zero,X	ASL, op,X	16	2	6
ass	ASL op	0E	3	6
ass,X	ASL op,X	1E	3	7

BCC
Salta se C=0

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BCC op	90	2	2**

BCS
Salta se C=1

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BCS op	B0	2	2**

BEQ
Salta se Z=1

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BEQ op	F0	2	2**

BIT
A ∧ M, M7→N, M6→V

N	Z	C	I	D	V
M7	m	/	/	/	M6

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
pag zero	BIT op	24	2	3
ass	BIT op	2C	3	4

BMI
Salta se N=1

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BMI op	30	2	2**

BNE
Salta se Z=0

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BNE op	D0	2	2**

BPL
Salta se N=0

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BPL op	10	2	2**

BRK
Interrupt; PC+2↓P↓

N	Z	C	I	D	V
/	/	/	1	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	BRK	00	1	7

Nota: il flag B viene messo a 1 prima di memorizzare il registro P nell'area stack.

BVC
Salta se V=0

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BVC op	50	2	2**

BVS
Salta se V=1

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BVS op	70	2	2**

CLC
0→C

N	Z	C	I	D	V
/	/	0	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	CLC	18	1	2

CLD
0→D

N	Z	C	I	D	V
/	/	/	/	0	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	CLD	D8	1	2

CLI
0→I

N	Z	C	I	D	V
/	/	/	0	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	CLI	58	1	2

CLV
0→V

N	Z	C	I	D	V
/	/	/	/	/	0

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	CLV	B8	1	2

CMP
A-M

N	Z	C	I	D	V
m	m	m	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	CMP #op	C9	2	2
pag zero	CMP op	C5	2	3
pag zero,X	CMP op,X	D5	2	4
ass	CMP op	CD	3	4
ass,X	CMP op,X	DD	3	4*
ass,Y	CMP op,Y	D9	3	4*
(ind,X)	CMP (op,X)	C1	2	6
(ind),Y	CMP (op),Y	D1	2	5*

CPX
X-M

N	Z	C	I	D	V
m	m	m	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	CPX #op	E0	2	2
pag zero	CPX op	E4	2	3
ass	CPX op	EC	3	4

CPY
Y-M

N	Z	C	I	D	V
m	m	m	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	CPY #op	C0	2	2
pag zero	CPY op	C4	2	3
ass	CPY op	CC	3	4

DEC
M-1→M

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
pag zero	DEC op	C6	2	5
pag zero,X	DEC op,X	D6	2	6
ass	DEC op	CE	3	6
ass,X	DEC op,X	DE	3	7

DEX
X-1→X

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	DEX	CA	1	2

DEY
Y-1→Y

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	DEY	88	1	2

EOR
A∀M→A

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	EOR #op	49	2	2
pag. zero	EOR op	45	2	3
pag. zero,X	EOR op,X	55	2	4
ass	EOR op	4D	3	4
ass,X	EOR op,X	5D	3	4*
ass,Y	EOR op,Y	59	3	4*
(ind,X)	EOR (op,X)	41	2	6
(ind),Y	EOR (op),Y	51	2	5*

INC
M+1→M

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
pag zero	INC op	E6	2	5
pag zero,X	INC op,X	F6	2	6
ass	INC op	EE	3	6
ass,X	INC op,X	FE	3	7

INX
X+1→X

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	INX	E8	1	2

INY
Y+1→Y

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	INY	C8	1	2

JMP
(PC+1)→PCL
(PC+2)→PCH

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
ass	JMP op	4C	3	3
ind	JMP (op)	6C	3	5

JSR
PC+1 ↓, (PC+1)→PCL
(PC+2)→PCH

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
ass	JSR op	20	3	6

LDA
M→A

N	Z	C	I	D	V
m	m	m	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	LDA #op	A9	2	2
pag zero	LDA op	A5	2	3
pag zero,X	LDA op,X	B5	2	4
ass	LDA op	AD	3	4
ass,X	LDA op,X	BD	3	4*
ass,Y	LDA op,Y	B9	3	4*
(ind,X)	LDA (op,X)	A1	2	6
(ind),Y	LDA (op),Y	B1	2	5*

LDX
M→X

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	LDX #op	A2	2	2
pag zero	LDX op	A6	2	3
pag zero,Y	LDX op,Y	B6	2	4
ass	LDX op	AE	3	4
ass,Y	LDX op,Y	BE	3	4*

LDY
M→Y

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	LDY #op	A0	2	2
pag zero	LDY op	A4	2	3
pag zero,X	LDY op,X	B4	2	4
ass	LDY op	AC	3	4
ass,X	LDY op,X	BC	3	4*

LSR

N	Z	C	I	D	V
m	m	m	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
acc	LSR	4A	1	2
pag zero	LSR op	46	2	5
pag zero,X	LSR op,X	56	2	6
ass	LSR op	4E	3	6
ass,X	LSR op,X	5E	3	7

NOP
Nessuna operazione

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	NOP	EA	1	2

ORA
A ∨ M→A

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	ORA #op	09	2	2
pag zero	ORA op	05	2	3
pag zero,X	ORA op,X	15	2	4
ass	ORA op	0D	3	4
ass,X	ORA op,X	1D	3	4*
ass,Y	ORA op,Y	19	3	4*
(ind,X)	ORA (op,X)	01	2	6
(ind),Y	ORA (op),Y	11	2	5

PHA
A↓

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	PHA	48	1	3

PHP
P↓

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	PHP	08	1	3

PLA
A↑

N	Z	C	I	D	V
m	m	/	/	/	/

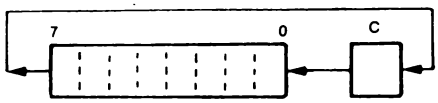
MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	PLA	68	1	4

PLP
P↑

N	Z	C	I	D	V
dallo stack					

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	PLP	28	1	4

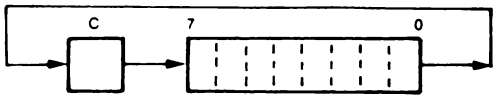
ROL



N	Z	C	I	D	V
m	m	m	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
acc	ROL	2A	1	2
pag zero	ROL op	26	2	5
pag. zero,X	ROL op,X	36	2	6
ass	ROL op	2E	3	6
ass,X	ROL op,X	3E	3	7

ROR



N Z C I D V
m m m / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
acc	ROR	6A	1	2
pag zero	ROR op	66	2	5
pag zero,X	ROR op,X	76	2	6
ass	ROR op	6E	3	6
ass,X	ROR,op	3	7	

RTI
P↑PC↑

N Z C I D V
dallo stack

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	RTI	40	1	6

RTS
PC↑, PC+1→PC

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	RTS	60	1	6

SBC
A-M-C→A
C=riporto

N Z C I D V
m m m / / m

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	SBC #op	E9	2	2
pag zero	SBC op	E5	2	3
pag. zero,X	SBC op,X	F5	2	4
ass	SBC op	ED	3	4
ass,X	SBC op,X	FD	3	4*
ass,Y	SBC op,Y	F9	3	4*
(ind,X)	SBC (op,X)	E1	2	6
(ind),Y	SBC (op,)Y	F1	2	5*

SEC
1→C

N	Z	C	I	D	V
/	/	1	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	SEC	38	1	2

SED
1→D

N	Z	C	I	D	V
/	/	/	/	1	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	SED	F8	1	2

SEI
1→I

N	Z	C	I	D	V
/	/	/	1	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	SEI	78	1	2

STA
A→M

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
pag zero	STA op	85	2	3
pag. zero,X	STA op,X	95	2	4
ass	STA op	8D	3	4
ass,X	STA op,X	9D	3	5
ass,Y	STA op,Y	99	3	5
(ind,X)	STA (op,X)	81	2	6
(ind),Y	STA (op,)Y	91	2	6

STX
X→M

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
pag. zero	STX op	86	2	3
pag. zero,Y	STX op,Y	96	2	4
ass	STX op	8E	3	4

STY
Y→M

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
pag zero	STY op	84	2	3
pag zero,X	STY op,X	94	2	4
ass	STY op	8C	3	4

TAX
A→X

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	TAX	AA	1	2

TAY
A→Y

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	TAY	A8	1	2

TSX
S→M

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	TSX	BA	1	2

TXA
X→A

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	TXA	8A	1	2

TXS
X→S

N	Z	C	I	D	V
/	/	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	TXS	9A	1	2

TYA
Y→A

N	Z	C	I	D	V
m	m	/	/	/	/

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	TYA	98	1	2

4.3 UTILIZZO DEL MONITOR

La disponibilità del comando MONITOR facilita la stesura e l'utilizzo di routine in linguaggio macchina e consente di indagare senza troppa fatica sul comportamento del sistema.

Per entrare in MONITOR basta scrivere la parola chiave e premere RETURN oppure premere SHIFT-F7. Appare sul video un messaggio di questo tipo:

```
MONITOR

    PC   SR  AC  XR  YR  SP
; FB000  00  00  00  00  F9
```

dove sotto la parola MONITOR le 6 sigle rapresentano i seguenti registri del processore:
- PC, registro contatore del programma (Program Counter);
- SR, registro di stato, che altrove indichiamo con P, contiene i flag;
- AC, registro accumulatore;
- XR, registro indice X;
- YR, registro indice Y;
- SP, registro puntatore all'area stack, che altrove indichiamo con S;

e nella linea seguente sono visualizzati i relativi contenuti.

L'entrata in MONITOR non disturba il programma BASIC eventualmente presente in memoria, e quando si utilizza il comando X per tornare al BASIC, si puó continuare a lavorare con esso.
Se in una routine in linguaggio macchina poniamo nel primo byte di un'istruzione il numero $00, codice operativo corrispondente all'istruzione BRK, quando esso viene incontrato la routine si interrompe, ma invece di ritornare il controllo al BASIC entra in MONITOR e viene visualizzato il relativo messaggio.

Il MONITOR riconosce gli indirizzi esadecimali come formati da 5 caratteri, infatti il primo carattere, da 0 a F, seleziona il banco. Gli indirizzi esadecimali si scrivono senza prefisso, ma si puó anche premettere ad essi il carattere $. Possiamo utilizzare numeri decimali premettendo ad essi il carattere + (più), ma in questo caso non possiamo indicare il banco.

Il MONITOR ci consente di convertire numeri con le seguenti basi:
- esadecimali, con prefisso $
- decimali, con prefisso +
- ottali, con prefisso &
- binari, con prefisso %

basta scrivere il numero e premere RETURN. Per esempio, scrivendo il primo numero, tra quelli che seguono, otteniamo il numero nelle 4 basi disponibili:

```
+65535
 $FFFF
 +65535
 &177777
 %1111111111111111
```

come si vede viene ripetuto anche il numero di partenza.

Il comando A consente di assemblare un programma linea dopo linea; si scrive A, seguito da un indirizzo e da una istruzione, il sistema riscrive la linea facendo seguire all'indirizzo il codice macchina prodotto e sposta verso destra il testo che è stato introdotto. Per esempio scriviamo:

```
A 01C00 LDY #$00
```

otteniamo:

```
A 01C00 A0 00          LDY #$00
A 01C02
```

cioè compare già l'indirizzo per l'istruzione seguente, e così via. Se scriviamo in sequenza quanto indicato nella prima colonna, vediamo visualizzato quello che compare nella seconda:

```
A 01C00 LDY #$00          A 01C00 A0 00      LDY #$00
A 01C02 STY $FF00         A 01C02 8C 00 FF   STY $FF00
A 01C05 STA $D600         A 01C05 8D 00 D6   STA $D600
A 01C08 ....              A 01C08 ....
```

A questo punto il codice macchina sta in memoria agli indirizzi indicati; per verificarlo possiamo chiedere con il comando M la visualizzazione della memoria da 01C00 a 01C08. Possiamo scriverlo in un file, su disco o su cassetta con il comando S, indicando l'indirizzo iniziale e quello finale.

Il comando C consente di confrontare due blocchi di memoria byte dopo byte; esso si scrive, per esempio:

C 00200 0021F 00400 0041F

ottenendo di confrontare tra loro i due blocchi del banco 0 di indirizzi $0200-$021F e $0400-$041F. Il MONITOR visualizza gli indirizzi dei byte del primo blocco che sono diversi dai corrispondenti del secondo blocco; se non viene visualizzato alcun indirizzo i due blocchi sono uguali.

Il comando D consente di disassemblare un blocco contenente del codice macchina; naturalmente bisogna dare come primo indirizzo quello del codice operativo di un'istruzione, altrimenti viene fuori un pasticcio. Per esempio, se scriviamo:

D FAFA2 FAFA4 otteniamo in risposta:

. FAFA2 4C EA 92 JMP $92EA

e con questa operazione abbiamo cercato l'indirizzo di inizio della routine che esegue l'operazione di *garbage collection* nella zona dinamica delle stringhe, il cui vettore per i salti indiretti inizia all'indirizzo $AFA2 del banco 15. Ora possiamo chiedere di disassemblare tutta la routine, che inizia all'indirizzo $92EA del banco 15, ma per lavorare meglio dobbiamo sapere l'indirizzo della sua ultima istruzione. Questo si puó ottenere utilizzando il comando H per ricercare a partire dall'indirizzo di inizio il primo RTS, cioè il primo codice esadecimale 60, e facendo qualche controllo, infatti la routine potrebbe contenere dei salti che la fanno proseguire oltre il primo RTS. Possiamo lavorare visualizzando sul video, ma rallentando lo scrolling con la pressione del tasto COMMODORE. Supponiamo che la routine termini all'indirizzo $93E0; diciamo supponiamo dato che non abbiamo studiato attentamente il listato, ma ci siamo limitati a scorrere gli indirizzi ed a disassemblare fino a un codice operativo RTS che sia oltre l'indirizzo di salto più alto incontrato. Non riportiamo il listato, che ogni lettore puó ottenere eseguendo i comandi suggeriti.

Il comando F serve per memorizzare in una serie consecutiva di byte un valore; si scrive, per esempio:

F 02000 0209F 00

per ottenere di scrivere $00 nei byte da $2000 a $209F del banco 0.

Il comando G fa eseguire una routine in linguaggio macchina che inizia all'indirizzo specificato. Scrivendo:

G 03000

si ottiene di far eseguire il programma in linguaggio macchina che deve essere stato preventivamente memorizzato in $3000 nel banco 0. Se si desidera che alla fine dell'esecuzione della routine il controllo torni al MONITOR, questa deve terminare con l'istruzione BRK invece di RTS.

Il comando H serve per cercare in un blocco di memoria dei dati, che possono essere espressi come valori numerici o come caratteri.

H F0000 FFFFF 'RESTOR

cerca in tutto il banco 15 la stringa RESTOR; si noti che la stringa deve essere preceduta dall'apice, ma non seguita anche dall'apice. Il MONITOR elenca gli indirizzi dei byte dove trova che inizia tale stringa.

H F0000 F2000 6D 4C 56

cerca nel blocco di memoria del banco 15 che inizia in $0000 e termina in $2000 i 3 byte che contengono $6D, $4C e $56. Se non li trova non visualizza, altrimenti visualizza l'indirizzo del primo byte delle terne trovate uguali.

Il comando L serve per caricare in memoria un file binario a partire da un determinato indirizzo. Si scrive:

L "nome file" n indirizzo

dove 'n' è il numero logico della periferica (1 per la cassetta e 8 per il disco) e 'indirizzo' puó anche mancare. Se manca 'indirizzo' il file viene caricato a partire dallo stesso indirizzo nel quale si trovava al momento della sua memorizzazione; se esso è presente il file viene caricato all'indirizzo indicato.
Si deve prestare particolare attenzione prima di modificare l'indirizzo di caricamento di una routine in linguaggio macchina, dal momento che essa puó contenere indirizzi assoluti (come JMP indirizzo) e quindi non funzionare più se spostata in memoria.

Il comando R mostra gli stessi registri del messaggio iniziale del MONITOR. Se dopo la visualizzazione dei registri si porta il cursore sul contenuto di qualcuno di essi e si modifica il valore, quando si preme RETURN viene memorizzato il nuovo valore.
Con questo sistema si puó modificare il contenuto del registro PC, ponendovi l'indirizzo di inizio di una routine, e poi usare il comando G senza indirizzo per eseguirla.

Il comando S si utilizza per memorizzare in un file binario blocchi di memoria; si scrive:

S "nome file" n ind1 ind2

dove 'n' è il numero logico della periferica, 'ind1' e 'ind2' sono gli indirizzi iniziali e finali del blocco di memoria. L'indirizzo 'ind2' deve essere superiore di 1 all'indirizzo dell'ultimo byte da salvare.

Il comando T trasferisce il contenuto di un blocco di memoria in un altro posto lasciandola invariata nella vecchia posizione, cioè esegue una copia. Si scrive:

T ind1v ind2v indn

dove 'ind1v' e 'ind2v' sono gli indirizzi iniziali e finali del blocco da copiare e 'indn' è l'indirizzo iniziale del blocco dove eseguire la copia.

Il comando V è utilizzato per confrontare il contenuto di un file binario con il contenuto di un blocco di memoria; si scrive:

V "nome file" n indirizzo

dove 'n' indica il numero logico della periferica e 'indirizzo' è l'indirizzo iniziale del blocco di memoria da confrontare. Il confronto continua fino alla fine del file; se non viene riscontrata uguaglianza viene segnalato 'VERIFYING ERROR'.

Il comando @ serve per controllare lo stato del disco, per inviare comandi o per visualizzare la DIRECTORY; si scrive:

@ per visualizzare lo stato del disco
@,I per inizializzare il disco
@,$ per visualizzare la DIRECTORY.

4.4 LE ROUTINE DEL SISTEMA E IL LORO UTILIZZO

Nel paragrafo 3.2 abbiamo riportato alcune mappe della memoria, sono stati elencati gli indirizzi di inizio di alcune routine del sistema e i vettori per i salti indiretti. Per alcune routine abbiamo aggiunto alcune notizie a noi note, ma esula dagli scopi di questo libro listare il contenuto della memoria ROM del COMMODORE 128. Come abbiamo mostrato nel precedente paragrafo con il MONITOR è possibile disassemblare le routine e studiarle; in tale modo si puó sapere quali sono i registri coinvolti e le eventuali altre routine richiamate.

L'utente puó utilizzare le routine del sistema richiamandole direttamente, ma puó anche intervenire a modificarne il comportamento, utilizzando il vettore dei salti indiretti che si trova nella memoria RAM.

Abbiamo preparato un programma in BASIC, che abbiamo chiamato MUSIC1, il quale utilizza due routine in linguaggio macchina, INTER e SUONA, per ottenere una musichetta di fondo mentre il calcolatore lavora.
Nel calcolatore viene generata 60 volte al secondo una interruzione, che manda in esecuzione la routine IRQ di servizio dell'interruzione. Tale routine inizia all'indirizzo 64101 ($FA65) e viene richiamata dal sistema tramite il vettore dei salti indiretti situato nelle locazioni 788 e 789 ($0314 e $0315), che contengono l'indirizzo di inizio della routine (prima il byte LOW e poi il byte HIGH).
Noi possiamo intercettare la routine IRQ al suo inizio modificando l'indirizzo contenuto nei due byte $0314 e $0315, in modo che quando avviene l'interruzione venga eseguita una nostra routine, la quale peró deve terminare con l'istruzione:

JMP $FA65

per far eseguire la normale routine di interrupt.

La routine INTER deve svolgere almeno le seguenti funzioni:

– maschera le interruzioni (istruzione SEI);
– cambia il contenuto dei due byte $0314 e $0315, ponendovi l'indirizzo di inizio della routine SUONA;
– abilita le interruzioni (istruzione CLI) e termina.

Dopo l'esecuzione di INTER, quando scatta un'interruzione il sistema va ad esegui-

re la routine situata all'indirizzo che trova in $0314 e $0315; in conseguenza a tale indirizzo deve essere stata memorizzata la routine SUONA, che deve generare la musichetta. La routine SUONA deve essere costruita in modo da svolgere le sua azione sessanta volte al secondo.
Per avviare questo meccanismo viene eseguito una volta il programma MUSIC, il quale carica in memoria le due routine INTER e SUONA, prepara alcuni dati che servono per far funzionare il SID e termina. Dopo l'esecuzione di MUSIC1 il calcolatore suona mentre si possono svolgere con esso tutti gli altri lavori.

Noi abbiamo deciso di utilizzare per le due routine in linguaggio macchina e per le memorie di lavoro necessarie il buffer della cassetta, quindi non dobbiamo modificare i puntatori del BASIC.

La nostra musichetta si compone di 48 note, che indichiamo facendo seguire alla sigla della nota e all'eventuale carattere '#', che indica il *diesis*, un numero che indica l'ottava utilizzata:

FA2 LA2 DO3 RE3 RE#3 RE3 DO3 LA2
FA2 LA2 DO3 RE3 RE#3 RE3 DO3 LA2
LA#1 RE2 FA2 SOL2 SOL#2 SOL2 FA2 RE2
FA2 LA2 DO3 RE3 RE#3 RE3 DO3 LA2
DO2 MI2 SOL2 MI2 LA#1 RE#2 FA2 RE2
FA2 LA2 LA#2 SI2 DO3 DO2 DO3 DO2.

Le frequenze di queste note corrispondono a 48 numeri che dobbiamo dividere in byte HIGH e byte LOW per memorizzarli nei registri appositi del SID; essi sono quelli che seguono (abbiamo indicato la parte HIGH e la parte LOW separandole con una / e in notazione decimale):

5/207 7/81 8/180 9/196 10/89 9/196 8/180 7/81
5/207 7/81 8/180 9/196 10/89 9/196 8/180 7/81
3/224 4/226 5/207 6/133 6/232 6/133 5/207 4/226
5/207 7/81 8/180 9/196 10/89 9/196 8/180 7/81
4/90 5/123 6/133 5/123 3/224 5/44 5/207 4/226
5/207 7/81 7/193 8/55 8/180 4/90 8/180 4/90.

Per suonare utilizziamo i seguenti registri del SID (si ricordi che i registri sono per buona parte a sola scrittura e alcuni a sola lettura):

- 54272 ($D400), byte LOW frequenza voce 1;
- 54273 ($D401), byte HIGH frequenza voce 1;
- 54274 ($D402), byte LOW larghezza onda impulsiva voce 1;
- 54275 ($D403), bit 3-0 nibble HIGH del registro 54274;
- 54276 ($D404), bit 6 a 1 per selezionare onda impulsiva,
  bit 0, GATE BIT, a 1 per la nota, a 0 per l'intervallo;
- 54277 ($D405), valore ATTACK e DECAY per voce 1;
- 54278 ($D406), valore SUSTAIN e RELEASE per voce 1;
- 54296 ($D418), bit 3-0 per volume.

Inoltre abbiamo bisogno delle seguenti memorie di lavoro:

- 48 byte per la parte LOW della frequenza delle note, che memorizziamo da 2816 a 2863 ($0B00-$0B2F);

– 48 byte per la parte HIGH della frequenza delle note, che memorizziamo da 2870 a 2917 ($0B36-$0B65);
– 1 byte come flag nota/intervallo in 2864 ($0B30);
– 1 byte come contatore della durata della nota in 2865 ($0B31);
– 1 byte come puntatore alla nota in 2866 ($0B32);
– 1 byte come contatore dell'intervallo tra due note in 2867 ($0B33).

La routine INTER viene memorizzata da 2920 a 2937 ($0B68-$0B79) e la routine SUONA da 2939 a 3016 ($0B78-$0BCE).

La routine INTER pone l'indirizzo di inizio di SUONA nei byte $0314 e $0315 e memorizza 255 ($FF) nel flag $0B30. Segue il listato.

```
; FB000 00 00 00 00 F8

. 00B68  78        SEI
. 00B69  A9 7B     LDA #$7B
. 00B6B  8D 14 03  STA $0314
. 00B6E  A9 0B     LDA #$0B
. 00B70  8D 15 03  STA $0315
. 00B73  A9 FF     LDA #$FF
. 00B75  8D 30 0B  STA $0B30
. 00B78  58        CLI
. 00B79  60        RTS
```

Il programma MUSIC1 esegue le seguenti operazioni:
– carica da disco le due routine INTER e SUONA;
– memorizza nelle zone indicate le parti LOW e HIGH delle frequenze delle note, che sono registrate in due blocchi di linee DATA;
– memorizza nei registri del SID i valori definiti;
– lancia la routine INTER e termina.

```
1 REM MUSIC1
3 REM CARICA CODICE MACCHINA
5 REM IN ZONA BUFFER CASSETTA
7 BLOAD "INTER",D0,U8,P2920
9 BLOAD "SUONA",D0,U8,P2939
11 REM CARICA PARTE LOW FREQUENZE
13 FOR K=0 TO 47:READ X: POKE 2816+K,X:NEXT K
15 REM CARICA PARTE HIGH FREQUENZE
17 FOR K=0 TO 47:READ X: POKE 2870+K,X:NEXT K
19 REM SISTEMA MEMORIE DI LAVORO
21 POKE 2865,35:POKE 2866,1:POKE 2867,32
23 REM SISTEMA REGISTRI SID
25 BANK 15:POKE 54296,15
27 POKE 54276,65:REM ONDA RETTANGOLARE
29 POKE 54277,0:POKE 54278,197
31 POKE 54274,119:POKE 54275,2
33 REM ESEGUE ROUTINE INTER
```

```
35 BANK 15: SYS 2920
37 PRINT "PROGRAMMA TERMINATO"
39 STOP
41 REM DATI MUSICALI
43 DATA 207,81,180,196,89,196,180,81
45 DATA 207,81,180,196,89,196,180,81
47 DATA 224,226,207,133,232,133,207,226
49 DATA 207,81,180,196,89,196,180,81
51 DATA 90,123,133,123,224,44,207,226
53 DATA 207,81,193,55,180,90,180,90
55 DATA 5,7,8,9,10,9,8,7
57 DATA 5,7,8,9,10,9,8,7
59 DATA 3,4,5,6,6,6,5,4
61 DATA 5,7,8,9,10,9,8,7
63 DATA 4,5,6,5,3,5,5,4
65 DATA 5,7,7,8,8,4,8,4
```

Nella figura 4.19 è indicato lo schema logico della routine SUONA, riportando vicino ad ogni blocco l'indirizzo esadecimale del codice corrispondente nel listato.

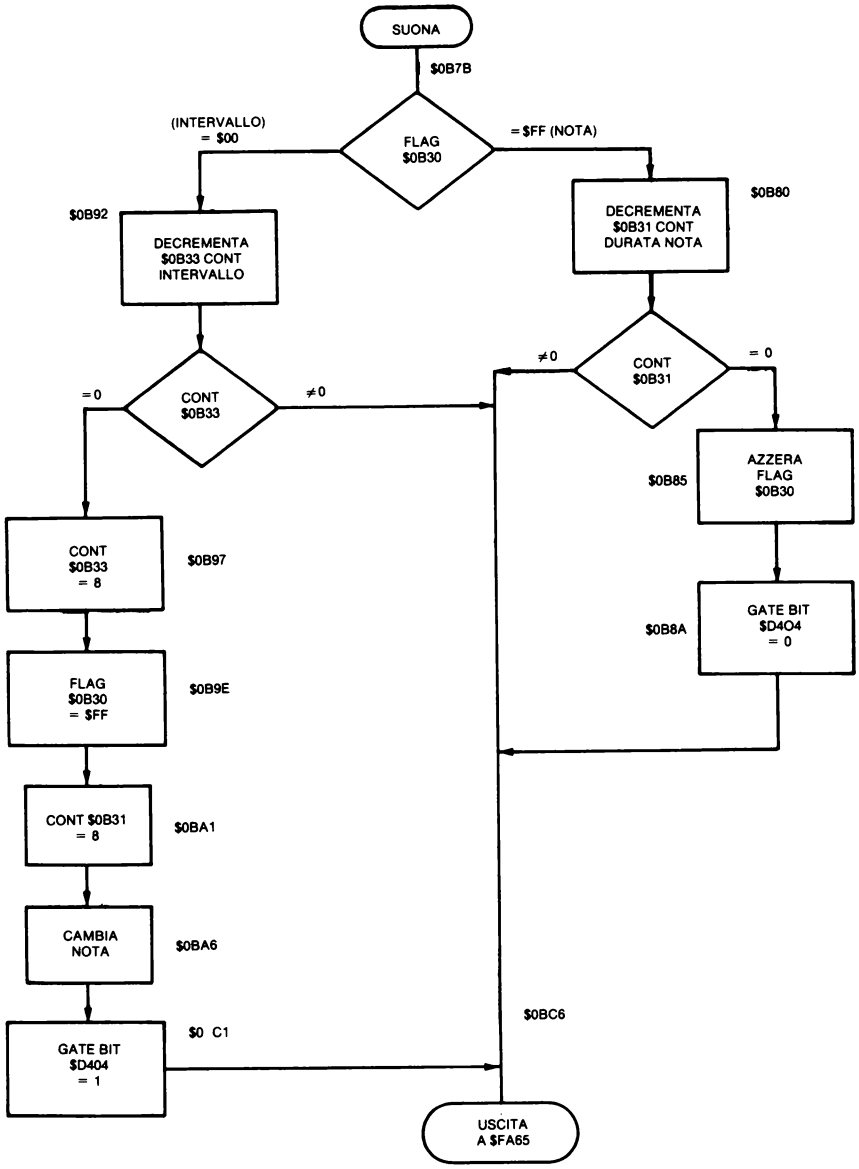


Fig. 4.19 Schema logico routine SUONA

Segue il listato di SUONA.

```
. 00B7B  AD 30 0B LDA $0B30
. 00B7E  F0 12    BEQ $0B92
. 00B80  CE 31 0B DEC $0B31
. 00B83  D0 41    BNE $0BC6
. 00B85  A9 00    LDA #$00
. 00B87  8D 30 0B STA $0B30
. 00B8A  A9 40    LDA #$40
. 00B8C  8D 04 D4 STA $D404
. 00B8F  4C 65 FA JMP $FA65
. 00B92  CE 33 0B DEC $0B33
. 00B95  D0 2F    BNE $0BC6
. 00B97  A9 08    LDA #$08
. 00B99  8D 33 0B STA $0B33
. 00B9C  A9 FF    LDA #$FF
. 00B9E  8D 30 0B STA $0B30
. 00BA1  A9 08    LDA #$08
. 00BA3  8D 31 0B STA $0B31
. 00BA6  AC 32 0B LDY $0B32
. 00BA9  EE 32 0B INC $0B32
. 00BAC  C0 30    CPY #$30
. 00BAE  D0 05    BNE $0BB5
. 00BB0  A9 01    LDA #$01
. 00BB2  8D 32 0B STA $0B32
. 00BB5  B9 FF 0A LDA $0AFF,Y
. 00BB8  8D 00 D4 STA $D400
. 00BBB  B9 35 0B LDA $0B35,Y
. 00BBE  8D 01 D4 STA $D401
. 00BC1  A9 41    LDA #$41
. 00BC3  8D 04 D4 STA $D404
. 00BC6  4C 65 FA JMP $FA65
```

Volendo modificare la qualità del suono si possono variare i parametri dell'onda rettangolare caricati dal programma in BASIC alla linea 31. Oppure si puó scegliere un'altra forma d'onda variando sia il valore 65 alla linea 27 del programma MUSIC1, sia i byte $0B8B (contenuto attuale #$40) e $0BC2 (contenuto attuale #$41) nei valori:

per onda triangolare
linea 27, porre 17
byte $0B8B, porre $10
byte $0BC2, porre $11

per onda a dente di sega
linea 27, porre 33
byte $0B8B, porre $20
byte $0BC2, porre $21.

4.5 LE NOTE MUSICALI

Riportiamo i valori da utilizzare come byte LOW e byte HIGH nel registro delle frequenze per ottenere una nota per 8 ottave.

Nota	Freq.	Valore	HIGH	LOW	
DO	16.4	278	1	22	ottava 0
DO#	17.3	295	1	39	
RE	18.4	313	1	57	
RE#	19.4	331	1	75	
MI	20.6	351	1	95	
FA	21.8	372	1	116	
FA#	23.1	394	1	138	
SOL	24.5	417	1	161	
SOL#	26	442	1	186	
LA	27.5	468	1	212	
LA#	29.1	496	1	240	
SI	30.9	526	2	14	
DO	32.7	557	2	45	ottava 1
DO#	34.6	590	2	78	
RE	36.7	625	2	113	
RE#	38.9	662	2	150	
MI	41.2	702	2	190	
FA	43.7	743	2	231	
FA#	46.2	788	3	20	
SOL	49	834	3	66	
SOL#	51.9	884	3	116	
LA	55	937	3	169	
LA#	58.3	992	3	224	
SI	61.7	1051	4	27	
DO	65.4	1114	4	90	ottava 2
DO#	69.3	1180	4	156	
RE	73.4	1250	4	226	
RE#	77.8	1324	5	44	
MI	82.4	1403	5	123	
FA	87.3	1487	5	207	
FA#	92.5	1575	6	39	
SOL	98	1669	6	133	
SOL#	103.8	1768	6	232	
LA	110	1873	7	81	
LA#	116.5	1985	7	193	
SI	123.5	2103	8	55	
DO	130.8	2228	8	180	ottava 3
DO#	138.6	2360	9	56	
RE	146.8	2500	9	196	
RE#	155.6	2649	10	89	
MI	164.8	2807	10	247	
FA	174.6	2973	11	157	
FA#	185	3150	12	78	
SOL	196	3338	13	10	
SOL#	207.7	3536	13	208	

LA	220	3746	14	162	
LA#	233.1	3969	15	129	
SI	246.9	4205	16	109	
DO	261.6	4455	17	103	ottava 4
DO#	277.2	4720	18	112	
RE	293.7	5001	19	137	
RE#	311.1	5298	20	178	
MI	329.6	5613	21	237	
FA	349.2	5947	23	59	
FA#	370	6300	24	156	
SOL	392	6675	26	19	
SOL#	415.3	7072	27	160	
LA	440	7493	29	69	
LA#	466.2	7938	31	2	
SI	493.9	8410	32	218	
DO	523.3	8910	34	206	ottava 5
DO#	554.4	9440	36	224	
RE	587.3	10001	39	17	
RE#	622.3	10596	41	100	
MI	659.3	11226	43	218	
FA	698.5	11894	46	118	
FA#	740	12601	49	57	
SOL	784	13350	52	38	
SOL#	830.6	14144	55	64	
LA	880	14985	58	137	
LA#	932.3	15876	62	4	
SI	987.8	16820	65	180	
DO	1046.5	17820	69	156	ottava 6
DO#	1108.7	18880	73	192	
RE	1174.7	20003	78	35	
RE#	1244.5	21192	82	200	
MI	1318.5	22452	87	180	
FA	1396.9	23787	92	235	
FA#	1480	25202	98	114	
SOL	1568	26700	104	76	
SOL#	1661.2	28288	110	128	
LA	1760	29970	117	18	
LA#	1864.7	31752	124	8	
SI	1975.5	33640	131	104	
DO	2093	35641	139	57	ottava 7
DO#	2217.5	37760	147	128	
RE	2349.3	40005	156	69	
RE#	2489	42384	165	144	
MI	2637	44904	175	104	
FA	2793.8	47574	185	214	
FA#	2960	50403	196	227	
SOL	3136	53400	208	152	
SOL#	3322.4	56576	221	0	
LA	3520	59940	234	36	
LA#	3729.3	63504	248	16	

CAPITOLO 5

IL MODO CP/M

5.1 INTRODUZIONE

Nella figura 3.3 del capitolo 3 è riportato lo schema di utilizzo della memoria con il Sistema Operativo CP/M (Control Program for Microcomputer), che è usato su molti microcalcolatori. Il significato delle sigle è il seguente:
– BDOS (Basic Disk Operating System) è la parte di sistema che deve risiedere sempre in memoria; contiene le routine necessarie alla gestione;
– BIOS (Basic Input Output System) è la parte che contiene le routine di input/output per comunicare con la tastiera, il video e il disco; deve risiedere sempre in memoria (il CP/M non puó essere usato senza disporre di un'unità a disco);
– CCP (Console Command Processor) è un programma interattivo, che deve risiedere sempre in memoria e consente il colloquio con l'utente e l'interpretazione dei comandi;
– TPA (Transient Program Area) è la zona di memoria utilizzata per la parte transiente (non residente) delle routine del sistema e per i programmi dell'utente.
Come si vede il blocco 0 della memoria RAM è utilizzato completamente per il sistema e contiene la mappa del video; i programmi utente vanno nel banco 1 della memoria RAM. Anche nel modo CP/M vengono sfruttate le possibilità di gestione dei banchi di memoria del COMMODORE 128.

Appare subito come sia diversa l'impostazione del sistema rispetto al modo 128. In questo caso vengono tenute in memoria solo le routine che risultano necessarie, mentre le altre sono caricate da disco quando servono.

Nel modo CP/M, inoltre, il sistema utilizza il processore Z80, quindi per programmare in linguaggio macchina è necessario studiare il linguaggio macchina di quel microprocessore.
Noi riteniamo che, date le grandi possibilità offerte dal COMMODORE 128 per programmare in modo 128 sia in BASIC che in ASSEMBLER, l'utente preferirà programmare in tale modo, mentre ricorrerà al CP/M per utilizzare molti interessanti e utili pacchetti di programmi disponibili sul mercato. Abbiamo provato l'interprete MBASIC della Microsoft con soddisfazione, ma il messaggio iniziale di partenza del BASIC segnala solo un po' più di 35000 byte disponibili, che sono sempre abbastanza, ma sicuramente molto meno di quelli disponibili nel modo 128.

Qualora un utente sia interessato ad apprendere i dettagli di funzionamento del sistema operativo CP/M, egli dovrà procurarsi un manuale completo del sistema e studiarselo. Potrà anche acquistare un interprete BASIC o compilatori per altri linguaggi, o anche un assemblatore, da utilizzare in CPM, e studiarne i relativi manuali. Inoltre in modo CP/M è disponibile anche un compilatore BASIC.

5.2 I PACCHETTI INTEGRATI PERFECT FILER, WRITER E CALC

Abbiamo potuto provare i due pacchetti Perfect Filer e Perfect Writer (della Perfect Software, Inc.), che risultano fra loro integrati, e abbiamo letto sui relativi manuali che essi sono integrati anche con il pacchetto Perfect Calc, che purtroppo non siamo riusciti ad avere.
I tre pacchetti funzionano in CP/M e sono stati personalizzati per il COMMODORE 128.
Per poter utilizzare questi pacchetti è necessario disporre di una unità a disco e di una stampante.

Perfect Filer è un pacchetto per gestire basi di dati, cioè archivi di dati. Esso consta in un voluminoso manuale, al momento in inglese, e tre floppy, che contengono i programmi, di cui uno registrato sulle due facce.

Perfect Writer è un pacchetto per la scrittura dei testi (word processor). Esso consta in un voluminoso manuale, anch'esso attualmente in inglese, e tre floppy, di cui due registrati sulle due facce.

Perfect Calc è un pacchetto del tipo *foglio elettronico*, cioè consente di preparare modelli di calcoli e di eseguirli, predisponendo i dati in modo tabellare.

Ci sembra opportuno spendere alcune parole sull'atteggiamento che l'utente dovrebbe avere verso i pacchetti di software, che si stanno diffondendo sul mercato in modo sempre più ampio. I pacchetti sono formati da molti programmi, ognuno con funzioni specifiche, ed è importante capire a cosa serve ogni programma e come si usa. Questo puó essere compreso da chiunque, anche da persone che non sanno e non desiderano imparare a programmare un calcolatore.
Per affrontare bene l'argomento è necessario avere almeno un'idea generale delle prestazioni fornite dal pacchetto e leggere con molta attenzione il manuale. In generale questi manuali contengono una parte introduttiva che consente di cominciare a prendere confidenza con il pacchetto. Viene raccomandato di non cominciare ad utilizzare i dischetti senza averne prima fatto una copia, per motivi di sicurezza. Le prime cose da fare sono le seguenti:
– applicare le etichette protettive sulle finestre laterali delle buste dei dischetti;
– listare le directory con il comando DIR del CP/M, dopo aver attivato la stampante con CNTRL-P;
– formattare con il comando FORMAT del CP/M un numero di dischetti sufficienti per eseguire le copie;
– utilizzare il programma PIP del CP/M per eseguire le copie di tutti i dischetti;
– listare le directory dei dischetti copiati e confrontarle con quelle dei dischetti originali (se non corrispondono rifare le copie);
– conservare i dischetti originali in luogo sicuro e utilizzare le copie per lavorare.

Vediamo una sintesi di quello che si puó fare con Perfect Filer.
Dopo aver definito il tracciato del record, aver organizzato i dati che interessano in gruppi di dati significativi, ognuno composto da campi, ed aver fissato le dimensioni di ogni campo, si crea la maschera video da utilizzare per l'introduzione dei dati. Tale maschera viene facilmente composta sul video, assegnando un nome distintivo a ogni gruppo e a ogni campo e usando il carattere *underscore* il numero di volte necessario per indicare il numero dei caratteri di ogni campo.
Il lavoro procede facendo comparire in una finestra video sovrapposta al testo il menu dei comandi disponibili ad ogni passo, con la possibilità di scegliere rapida-

mente quello che si vuole fare. La maschera video è costantemente visualizzata e si puó corregerla se non va bene o se si è dimenticato qualcosa. Quando si è sicuri che la maschera va bene, essa viene memorizzata su disco assegnando un nome al file di cui la maschera rappresenta il tracciato del record. Oltre ai nomi distintivi dei campi, che sono visualizzati nella maschera video, vengono chiesti anche i nomi da assegnare ai campi nel file memorizzato su disco; tali nomi servono per particolari lavori tipo selezioni di record o altro. I nomi dei campi per il file possono coincidere con quelli utilizzati nella maschera video o essere diversi. Il tracciato del record puó essere listato; tale lista va conservata come parte della documentazione relativa alla gestione dell'archivio che si sta creando.
Inoltre il tracciato del record puó essere corretto anche dopo averlo memorizzato su disco, basta richiamarlo, modificarlo e memorizzarlo nuovamente. A ogni campo è possibile assegnare delle opzioni particolari, tipo 'Yes/No', a significare che il campo puó essere presente o mancare.
Dopo aver definito le caratteristiche del file esso va realizzato, cioè si devono caricare i dati dei record. Questa operazione è abbastanza semplice, infatti compare sul video la maschera del record e, spostando il cursore nei vari campi per mezzo dei tasti appositi, si scrivono i contenuti con possibilità di correzione.
Al file possono essere aggiunti ed anche cancellati record e si possono modificare i record già esistenti. Da un file possono essere estratti gruppi di record aventi determinate caratteristiche comuni, con i quali generare altri file. I record possono essere ordinati in base al valore di alcuni campi, fino ad un massimo di 9.
E' possibile modificare il tracciato del record di un file anche dopo aver già caricato dati nel file, senza che questi vadano persi.
Si possono produrre diversi tipi di prospetti estraendo record dal file secondo predefiniti criteri di selezione.
Si possono trasferire dati a Perfect Calc, farli elaborare e avere indietro i risultati da aggiungere ai record di un file.
Si possono utilizzare alcuni campi dei record trasferendoli in documenti scritti con Perfect Writer, in modo da produrre documenti o lettere personalizzate.

Abbiamo fatto un cenno al problema della preparazione del tracciato record per un file nel capitolo 2 descrivendo il programma ES7. Quando si definisce la struttura di un file Perfect Filer crea una serie di file di servizio che contengono tutte le informazioni necessarie per gestire il file di dati effettivo; tali file hanno lo stesso nome e diversa estensione. Inoltre vengono utilizzati file di lavoro temporanei per poter realizzare le diverse fasi. Si raccomanda di cambiare il dischetto e memorizzare i file di dati su un dischetto diverso da quello dei programmi, soprattutto se si prevede che il file di dati possa avere molti record.

Ai record dei file creati con Filer si puó accedere per chiave, cioè in base al contenuto di un particolare campo. E' possibile sia l'accesso sequenziale che quello diretto, nel senso che si puó definire un qualunque criterio di selezione in base al quale estrarre record dal file.

Riportiamo ora una sintesi di quello che si puó fare con Perfect Writer. Le operazioni fondamentali che compie un programma elaboratore di testi sono:
– aprire un file di testo in memoria;
– scrivere il testo mentre viene visualizzato sul video, con possibilità di muoversi nel testo andando a inserire linee dove si desidera, spostando paragrafi, cancellando e modificando;
– inserire in un testo in scrittura parti che vengono caricate da altri file;
– salvare parti di testo su disco come file;

– dare al testo l'impaginazione voluta, con linee di una determinata lunghezza, indentazioni, allineamenti (giustificazioni);
– ricercare e correggere frasi o parole in tutto il testo o in una parte di esso;
– memorizzare i file di testo sul dischetto;
– stampare i testi nel modo desiderato.

Tutte queste cose si possono fare con il pacchetto che stiamo descrivendo, ma anche molte altre.
Perfect Writer lavora con il sistema di mostrare su richiesta il menu di scelta dei comandi possibili nella fase in atto, sovrapponendolo al video, e, inoltre, se si chiede aiuto (help) premendo il tasto con il '?', compare una spiegazione dei comandi del menu.
Esso consente di trattare file di testo molto lunghi, infatti provvede a memorizzare su disco una parte del testo, quando esso non puó più essere contenuto tutto in memoria, senza problemi per l'utente. Altra cosa notevole è la possibilità di gestire contemporaneamente in memoria più file di testo, fino a 7, in zone diverse della memoria, e di poter passare da uno a l'altro a seconda delle necessità. E' possibile dividere il video in due zone (due finestre) e visualizzare in ognuna di esse documenti diversi.

Per quanto riguarda la formattazione del testo con Perfect Writer si puó lavorare in due modi. Il primo consiste nel preparare sul video il testo nello stesso modo nel quale si desidera che venga stampato (si usa dire 'formattandolo sul video'), cioè stabilendo a priori la lunghezza della riga, inserendo linee bianche, indentazioni, tabulazioni, centrature di parti di testo, giustificazioni. In sostanza si vede sul video il testo come apparirà in stampa; esso viene memorizzato su disco nel formato creato e così viene stampato.
Il secondo modo consiste nel preparare prima con comandi appositi una o più maschere di formato per il testo e per parti di testo, assegnare loro un nome o un simbolo di riconoscimento e memorizzarle. Durante la scrittura del testo, che viene scritto più liberamente senza vedere il risultato finale, si devono inserire i richiami alle maschere; al momento della stampa provvede il sistema a applicare le maschere e a produrre il testo formattato.
Il primo metodo risulta più facile da applicare, il secondo puó essere utilizzato dopo aver fatto un po' di esperienza.

Per quanto riguarda il carattere di stampa, quello che si puó ottenere dipende dalla stampante collegata; il programma consente di usare il carattere normale, il grassetto e il corsivo, cioè di richiederlo utilizzando appositi caratteri di controllo, si tratta di vedere come questi vengono riconosciuti e utilizzati dalla stampante.

Una caratteristica molto interessante di Perfect Writer è quella di poter creare indici, utilizzando intestazioni particolari di parti di testo, come capitoli, paragrafi e sottoparagrafi. Inoltre esso consente di creare elenchi di parole, contrassegnate nel testo, in ordine alfabetico, indicando la pagina del testo dove esse compaiono; questa possibilità sarà particolarmente apprezzata da tutti coloro che hanno preparato almeno una volta un indice analitico per un libro.

Il pacchetto contiene anche il programma SPELLING, cioè un programma che puó analizzare, se richiesto, tutto un testo, confrontando le parole con quelle di un dizionario che è memorizzato su un dischetto, e correggendo quelle sbagliate. E' possibile creare un dizionario personale.

Altra possibilità è quella di poter utilizzare un particolare dizionario, chiamato THESAURUS, che è memorizzato su un dischetto; esso contiene i sinonimi di un gruppo di parole e l'utente puó richiamarlo mentre sta scrivendo. In caso di richiamo il video viene diviso a metà e compare l'elenco dei sinonimi per la parola richiesta, così l'utente puó trovare la parola giusta per evitare ripetizioni.

Vediamo ora come si deve procedere per scrivere una lettera con Perfect Writer e personalizzarla con i dati di un file preparato con Perfect Filer. Nel testo della lettera vanno inseriti tra i caratteri *minore* e *maggiore* i nomi dei campi dei dati del record da utilizzare; per esempio <nome>. Al momento della stampa si deve comunicare il nome del file da utilizzare per prelevare i dati; tale file potrà essere stato preparato con particolari criteri di selezione utilizzando le apposite opzioni del pacchetto Filer.

Perfect Writer puó comunicare con Calc e ottenere tabelle calcolate da inserire nei testi.

LISTINO LIBRI JACKSON

CODICE	TITOLO	PREZZO
	INFORMATICA: CONCETTI GENERALI	
511 A	COME PROGRAMMARE	15.000
500 H	DATA BASE: L'UTILE E IL DILETTEVOLE	15.000
503 A	PROGRAMMAZIONE STRUTTURATA, CORSO DI AUTOISTRUZIONE	15.000
101 H	TERMINI DELL'INFORMATICA E DELLE DISCIPLINE CONNESSE	50.000
539 A	LOGICA E DIAGRAMMI A BLOCCHI: TECNICHE DI PROGRAMMAZIONE	40.000
526 P	DATA BASE: CONCETTI E DISEGNO	22.500
GYS190	TRADUTTORI DI LINGUAGGI	26.000
545 P	VOI, L'AUTOMAZIONE E L'UFFICIO: 100 TAVOLE PER IL MANAGER	45.000
G 240	PAROLE BASE DELL'INFORMATICA	8.000
GYS245	CONCETTI DI INFORMATICA	43.000
GYS248	DATA PROCESSING	45.000
GY 264	DATA FILE TEORIA	50.000
GYS266	ARCHITETTURE DI SISTEMA	32.000
CI 126	I FONDAMENTI DELL'INFORMATICA	60.000
GY 354	SISTEMI INTELLIGENTI	28.000
CZ 419	ANALISI E PROGRAMMAZIONE	11.000
158 EC	INFORMATICA DI BASE I CONCETTI FONDAMENTALI HARDWARE E SOFTWARE	55.000
526 A	VOI E L'INFORMATICA	15.000
100 H	DIZIONARIO DI INFORMATICA	59.000
GY 551	I LINGUAGGI DELLA 4a GENERAZIONE	65.000
GYS552	PRIMA DEL LINGUAGGIO LA PROGRAMMAZIONE	35.000
GYS 559	C.S.P. - PROCESSI SEQUENZIALI	49.000
	INFORMATICA: SISTEMI OPERATIVI	
543 P	AMBIENTE UNIX	19.000
352 H	SISTEMI OPERATIVI PER MICROPROCESSORI VOL. 1	18.000
G 223	UNIX LA GRANDE GUIDA	70.000
353 H	SISTEMI OPERATIVI PER MICROPROCESSORI VOL. 2	18.000
G 237	SISTEMI OPERATIVI PER MICROPROCESSORI VOL. 3	18.000
GY 272	SISTEMI OPERATIVI PER MICROCOMPUTER	25.000
GY 273	MS-DOS LA GRANDE GUIDA	45.000
510 P	CP/M CON MP/M	29.000
CZ 538	MS DOS 2 E 3	49.000
G 543	XENIX	45.000
R 588	LAVORARE CON XENIX	70.000
GYS271	SISTEMI OPERATIVI	55.000
R 615	I COMANDI DI XENIX MAIL	12.500
092 D	SOFTWARE DI BASE E SISTEMI OPERATIVI	7.000
093 D	CP/M IL "SOFTWARE BUS"	7.000
094 D	MS-DOS E PC-DOS LO STANDARD IBM	7.000
009 H	UNIX	8.500
011 H	CP/M	8.500
012 H	PC - DOS	8.500
019 H	MS DOS	8.500
	INFORMATICA: LINGUAGGI	
501 A	IMPARIAMO IL PASCAL	16.000
502 A	INTRODUZIONE AL BASIC	25.000
500 P	PASCAL MANUALE E STANDARD DEL LINGUAGGIO	16.000

CODICE	TITOLO	PREZZO
329 A	PROGRAMMARE IN ASSEMBLER	14.000
513 A	PROGRAMMARE IN BASIC	8.000
512 P	SOLUZIONE DI PROBLEMI CON PASCAL	35.000
514 A	PROGRAMMARE IN PASCAL	19.000
516 A	INTRODUZIONE AL PASCAL	39.000
517 P	DAL FORTRAN IV AL FORTRAN 77 (II ED.)	32.000
521 A	50 ESERCIZI IN BASIC	17.000
525 A	BASIC PER TUTTI	23.000
534 A	MANUALE DEL BASIC	45.000
509 A	LOGO: POTENZA E SEMPLICITÀ	20.500
507 B	TUO PRIMO PROGRAMMA IN BASIC (II)	19.500
533 A	BASIC DALLA A ALLA Z	19.000
540 A	LINGUAGGIO ADA	19.500
541 P	LINGUAGGIO C	25.000
542 P	COBOL STRUTTURATO: CORSO DI AUTOISTRUZIONE	50.000
508 P	PROGRAMMARE IN C	39.000
G 233	COBOL PER MICROCOMPUTER	35.000
GYS246	ESERCIZI DI FORTRAN	20.000
GYS247	ESERCIZI IN PASCAL: ANALISI DEI PROBLEMI	29.000
GYS254	PROGRAMMAZIONE IN LINGUAGGIO ADA	42.000
GY 270	APL PER IL P.C. IBM	25.000
GYS274	DAL PASCAL AL MODULA 2	26.000
GYS311	LINGUAGGIO C IL LIBRO DELLE SOLUZIONI	24.000
GYS328	APPLICAZIONI IN PASCAL	32.000
GY 535	TURBO PASCAL	29.000
G 544	"Ç" LIBRARY	49.000
GYS550	PROLOG - LINGUAGGIO E APPLICAZIONE	32.000
R 589	TURBOPASCAL - LIBRERIA DI PROGRAMMI	45.000
042 T	LINGUAGGIO C	12.500
108 D	FORTH ANATOMIA DI UN LINGUAGGIO	7.000
107 D	FORTRAN E COBOL LINGUAGGI SEMPRE VERDI	7.000
086 D	ED È SUBITO BASIC VOL. 1	7.000
087 D	ED È SUBITO BASIC VOL. 2	7.000
034 T	PROLOG	14.000
035 T	LISP	12.500
001 H	COBOL	8.500
006 H	PASCAL	8.500
007 H	BASIC	8.500
010 H	FORTRAN 77	8.500
020 H	LOGO	8.500
022 H	FORTH	8.500
	INFORMATICA: LAVORO È SOCIETÀ	
519 P	COMPUTER GRAFICA	29.000
800 P	ODISSEA INFORMATICA	50.000
407 H	APPLICAZIONI DEL COMPUTER NELL'UFFICIO MODERNO	23.000
802 H	INFORMATICA MUSICALE	27.000
802 P	COMPUTERGRAPHIA	40.000
805 H	COMPUTER FEELINGS	20.000
806 P	COMPUTER PER L'INGEGNERIA EDILE	22.000
807 P	COMPUTER PER IL MEDICO	19.000
CI 231	COMPUTER IMAGE	40.000
CI 241	ODISSEA INFORMATICA STRATEGIE CULTURALI PER UNA SOCIETÀ INF.	32.000
G 400	COMPUTER GRAPHICS E ARCHITETTURA	27.000

GEJ 0006

CODICE	TITOLO	PREZZO
PV 409	COMPUTER GRAPHICS E MEDICINA	18.000
GY 487	MEDICO & COMPUTER	45.000
529 C	COMPUTER GRAPHICS	45.000
GY 548	INFORMATICA MEDICA	65.000
	INFORMATICA: SOFTWARE PACCHETTI APPLICATIVI	
556 H	VISICALC	24.000
570 P	CONTABILITÀ COL PERSONAL COMPUTER	27.000
525 P	WORDSTAR	24.000
546 P	MANUALE DEL DBASE II	24.000
578 P	PC NELL'ORG. DELLE PICCOLE AZIENDE: APPL. DEL MULTIPLAN	29.000
561 P	INTRODUZIONE AI FOGLI ELETTRONICI NELLA GESTIONE AZIENDALE	12.000
PP 219	LOTUS 1, 2, 3: GUIDA ITALIANA ALL'USO	21.000
G 234	RIORDINO E GESTIONE DEGLI ARCHIVI APPLICAZIONI CON PFS-FILE	30.000
PP 255	DBASE III GUIDA ITALIANA ALL'USO	45.000
PP 279	DBASE II CORSO DI ISTRUZIONE	47.000
PP 280	DBASE II CORSO AVANZATO DI ISTRUZIONE	60.000
PP 281	DBASE II CORSO COMPLETO D'ISTRUZIONE	90.000
PA 282	MODELLI DECISIONALI PER IL MANAGER	50.000
PA 288	PIANIFICAZIONE AZIENDALE PLANNING, MARKETING STRAT., BUDGETING	35.000
PP 310	LA GRANDE GUIDA LOTUS A SYMPHONY	70.000
PP 326	MULTIPLAN CORSO D'ISTRUZIONE	40.000
PP 344	FRAME WORK II - GUIDA ITALIANA ALL'USO	27.000
PP 351	WORD PROCESSING	27.000
PP 467	IMPARA 1.2.3. CON LA GRANDE GUIDA LOTUS	45.000
PP 468	CHART - CORSO ISTRUZIONE	45.000
PP 473	IL NUOVO 1-2-3 GUIDA ALL'USO DELLA VERSIONE ITALIANA 2 LOTUS 1-2-3	29.000
PA 474	BILANCIO, BUDGET, CASH FLOW (FLOPPY)	40.000
PP 475	DBASE III - CORSO DI PROGRAMMAZIONE	23.000
PA 476	PREVISIONE, PIANIFICAZIONE, SIMULAZIONE CON LOTUS 1-2-3 (FLOPPY)	60.000
PV 477	GUIDA ALLA BUSINESS GRAPHIC	20.000
PP 480	AUTOCAD	40.000
PP 481	RBASE 5000 - GUIDA ITALIANA ALL'USO	20.000
PP 537	IL MANUALE DI WINDOWS	60.000
PP 539	DBASE III - TECNICHE AVANZATE DI PROGRAMMAZIONE	42.000
PP 545	APPLICAZIONI DI DBASE III (FLOPPY)	50.000
PA 566	MODELLI DECISIONALI CON LOTUS 1-2-3 (FLOPPY)	40.000
PP 577	MANUALE DBASE III PLUS	49.000
039 T	WORDSTAR	12.500
040 T	LOTUS 1-2-3	12.500
043 T	WINDOWS	12.500
PP 621	I COMANDI DI DBASE III PLUS	12.500
095 D	GUIDA AI PACKAGE APPLICATIVI MERCEOLOGIA DEL SOFTWARE	7.000
096 D	VISICALC GUIDA RAPIDA ALL'UTILIZZO	7.000
098 D	WORD PROCESSING	7.000
103 D	LOTUS 1-2-3 E SIMPHONY IL FASCINO DELL'INTEGRAZIONE	7.000
104 D	DBASE II E III I PRINCIPI DI DATABASE	7.000
106 D	MULTIPLAN SPREADSHEET MULTISTRATO	7.000
110 D	PACKAGE A CONFRONTO PROVE DEI SOFTWARE PIÙ DIFFUSI	7.000
031 T	FRAMEWORK E FRAMEWORK II	12.500
033 T	MULTIPLAN 2.02	12.500
036 T	SYMPHONY	12.500
038 T	REFLEX	12.500
026 H	VISICALC	8.500

CODICE	TITOLO	PREZZO
027 H	EASY SCRIPT	8.500
032 H	WORD	8.500
033 H	PAGE MAKER	8.500
034 H	PROJECT	8.500
035 H	RBASE	8.500
	PERSONAL COMPUTER	
550 D	PROGRAMMI PRATICI IN BASIC	15.000
515 H	BASIC E LA GESTIONE DEI FILE VOL. I: METODI PRATICI	15.000
551 D	75 PROGRAMMI IN BASIC PER IL VOSTRO COMPUTER	12.000
552 D	PROGRAMMI DI MATEMATICA E STATISTICA IN BASIC	20.000
554 P	PROGRAMMI SCIENTIFICI IN PASCAL	29.000
516 H	BASIC E LA GESTIONE DEI FILE - VOL. 2	17.000
300 P	UNITÀ A DISCHI PER PERSONAL COMPUTER	15.000
CH 182	COMPUTER HARDWARE REALIZZ. PRATICHE PER GLI HC PIÙ DIFFUSI	18.000
CI 187	COMPUTER L'HOBBY E IL LAVORO	12.000
G 235	GRAFICA PER PERSONAL COMPUTER	39.000
GE 263	METODI DI INTERFACC. PERIFERICHE	43.000
GE 402	CORSO DI AUTOISTRUZIONE PER MICROCOMPUTER	35.000
PA 406	COME GESTIRE LA PICCOLA AZIENDA CON IL P.C.	22.000
PP 408	BUSINESS IN BASIC	23.000
CI 412	IL COMPUTER È UNA COSA SEMPLICE	15.000
CC 415	CONTROLLO DEI DISPOSITIVI DOMESTICI CON IL P.C.	23.000
CI 416	GRAFOLOGIA, NUMEROLOGIA, OROSCOPI	15.000
159 GC	PERSONAL COMPUTER DAL SOFTWARE DI BASE ALLE APPLICAZIONI D'UFFICIO	55.000
R 587	HARD DISK - LA GRANDE GUIDA	75.000
084 D	INTRODUZIONE AI PERSONAL COMPUTER VIVERE CON PC	7.000
099 D	SCRIVERE UN'AVVENTURA, 1000 AVVENTURE COL PROPRIO PC	7.000
100 D	GRAFICA E BASIC LE BASI DELLA COMPUTERGRAFICA	7.000
085 D	HARDWARE DI UN PERSONAL COMPUTER DENTRO E FUORI LA SCATOLA	7.000
101 D	GESTIONE DEI FILE IN BASIC E PASCAL VOL. 1	7.000
102 D	GESTIONE DEI FILE IN BASIC E PASCAL VOL. 2	7.000
113 D	DISEGNARE COL PERSONAL COMPUTER	7.000
105 D	PERSONAL E HOME COMPUTER A CONFRONTO	7.000
112 D	SUONO E MUSICA COL PERSONAL COMPUTER	7.000
109 D	COSTRUIRSI UN PERSONAL DATABASE	7.000
097 D	GUIDA ALL'ACQUISTO DI UN PERSONAL COMPUTER	7.000
088 D	TO DO OR NOT TO DO COME AVER CURA DEL PROPRIO PC	7.000
089 D	SOFTWARE STRUTTURATO CON ELEMENTI DI PASCAL	7.000
090 D	DIZIONARIO DI INFORMATICA	7.000
091 D	BASI DELLA PROGRAMMAZIONE STENDERE UN PROG. COME SI DEVE	7.000
004 H	PROGRAMMAZIONE	8.500
015 H	PROGRAMMI DI STATISTICA	8.500
	PERSONAL COMPUTER: COMMODORE	
347 D	VOI E IL VOSTRO COMMODORE 64	24.000
348 D	COMMODORE 64 - IL BASIC	28.000
400 D	FACILE GUIDA AL COMMODORE 64	13.500
400 B	COMMODORE 64 - FILE	19.000
409 B	COMMODORE 64 - LA GRAFICA E IL SUONO	34.000

GEJ 0006

CODICE	TITOLO	PREZZO
570 D	MATEMATICA E COMMODORE 64	26.500
350 D	LIBRO DEI GIOCHI DEL COMMODORE 64	24.000
573 D	GRAFICA E COMMODORE 64	15.000
575 D	TECNICHE DI PROGRAMMAZIONE SUL COMMODORE 64	16.500
572 D	LINGUAGGIO MACCHINA DEL COMMODORE 64 (FLOPPY)	35.000
413 B	COMMODORE 16 PER TE: BASIC 3.5	35.000
576 D	SISTEMA TOTOMAC: LA NUOVA FRONTIERA DEL TOTOCALCIO	29.000
548 B	64 PERSONAL COMPUTER E C64	45.000
427 B	C16 SEMPRE DI PIÙ	35.000
SDP222	STATISTICA AD UNA DIMENSIONE CON IL C64	24.000
CC 229	IMPARA IL BRIDGE CON IL COMPUTER: C64	50.000
CC 230	ROMANZO ROSA CON IL C64	40.000
CC 244	LAVORIAMO CON IL C16	20.000
CC 256	GUIDA AL COMMODORE PLUS 4	30.000
CC 260	AVVENTURE (COMMODORE 64)	20.000
CC 320	AMIGA HANDBOOK	35.000
CC 322	COMMODORE 128 OLTRE IL MANUALE	29.000
CC 323	PROGRAMMI PER COMMODORE 128	29.000
CC 324	PROGRAMMI PER C16	27.000
CC 329	LINGUAGGIO MACCHINA PER IL C16	16.000
CZ 541	128 E 64 - LE PERIFERICHE	32.000
CC 564	MANUALE RIPARAZIONE C64	55.000
CZ 532	MANUALE DI AMIGA	39.000
002 H	COMMODORE 64	8.500
005 H	VIC 20	8.500
	PERSONAL COMPUTER: SINCLAIR	
CC 286	SUPERBASIC PER SINCLAIR QL	30.000
CC 287	MANUALE DEL SINCLAIR QL	29.000
017 H	SINCLAIR SPECTRUM	8.500
	PERSONAL COMPUTER: IBM	
564 D	PROGRAMMI UTILI PER IBM PC	19.000
421 P	GUIDA AI PC IBM	21.000
G 217	GRAFICA PER IL PERSONAL COMPUTER IBM	39.000
CC 239	IMPARA IL BRIDGE CON IL COMPUTER IBM	50.000
GY 319	PC IBM MANUALE DEL LINGUAGGIO MACCHINA	45.000
GY 335	MAPPING PC IBM GESTIONE DELLA MEMORIA	42.000
PP 407	MANUALE BASE DEL PC IBM	22.000
041 T	PC IBM	12.500
	PERSONAL COMPUTER: OLIVETTI	
401 A	M20 LA PROGRAMMAZIONE BASIC PCOS	30.000
401 P	PRIMO LIBRO PER M24: MS DOS E GW BASIC	28.000
401 B	OLIVETTI M10: GUIDA ALL'USO	18.000
CL 216	BASIC IN 30 ORE PER M24 ED M20	32.000
CZ 483	MANUALE OLIVETTI M19	42.000
CZ 536	MANUALE PC 128 OLIVETTI PRODEST	29.000
CZ 582	PROGR. PER PC 128 OLIVETTI PRODEST (CASS.)	27.000
	PERSONAL COMPUTER: MSX	
CZ 181	30 PROGRAMMI PER MSX	20.000
417 D	MSX: IL BASIC	23.000
CC 261	AVVENTURE (MSX)	20.000
CC 289	SUPER PROGRAMMI PER MSX	35.000
CC 336	MSX LA GRAFICA	25.000

CODICE	TITOLO	PREZZO
111 D	STANDARD MSX	7.000
	PERSONAL COMPUTER: APPLE	
331 P	APPLE II GUIDA ALL'USO	31.000
416 P	MACINTOSH NEGLI AFFARI: MULTIPLAN E CHART	16.500
424 P	UN MAC PER AMICO: USO, APPLICAZIONI E PROGRAMMI PER MACINTOSH	12.000
PP 224	MACINTOSH ARTISTA: MACPAINT E MACDRAW	16.000
CCP277	APPLE IIC GUIDA ALL'USO	45.000
CC 312	PROGRAMMI PER APPLE IIC	13.000
CC 321	MICROSOFT BASIC PER APPLE MACINTOSH (VERS. 1.0 E 2.0)	24.000
CC 417	PROGRAMMI COMM. E FINANZIARI CON APPLE	22.000
CI 418	DISEGNI ANIMATI CON APPLE	22.000
CC 420	TECNICHE DI INTERFACCIAMENTO DELL'APPLE	20.000
340 H	APPLE MEMO	15.000
CC 576	IL MANUALE DELL'APPLE II GS	28.000
003 H	APPLE IIE IIC	8.500
	PERSONAL COMPUTER: ATARI - AMSTRAD - SHARP	
540 H	BASIC ATARI	18.000
CC 330	PROGRAMMI PER AMSTRAD CPC 464 CPC 664 - CPC 6128	29.000
CC 331	PROGRAMMI PER ATARI 130XE	19.000
CC 471	MANUALE ATARI 520 ST E 1040 ST	28.000
CC 486	WORD PROCESSING CON AMSTRAD PCW 8256/8512	35.000
032 T	AMSTRAD PCW 8256 e PCW 8512	14.000
014 H	SHARP MZ-80A	8.500
028 H	AMSTRAD 464 E 664	8.500
	COMUNICATION E TELEMATICA	
309 A	PRINCIPI E TECNICHE DI ELABORAZIONE DATI	20.000
518 D	TELEMATICA	28.000
528 P	TRASMISSIONE DATI	27.000
617 P	RETI DATI: CARATTERISTICHE, PROGETTO E SERVIZI TELEMATICI	40.000
GYS314	ELABORAZIONE DIGITALE DEI SEGNALI: TEORIA E PRATICA	25.000
PA 327	BANCHE DATI RICERCA ONLINE	26.000
158 LC	COMUNICAZIONI DALLE ONDE ELETTROMAGNETICHE ALLA TELEMATICA	55.000
CC 472	MODEM E PC USO E APPLICAZIONI	25.000
GTS478	RETI LOCALI	44.000
GTS479	IL MODEM - TEORIA, FUNZIONAMENTO	28.000
R 542	TRASMISSIONE DATI E PC	31.000
GT 555	LA TELEMATICA NELL'UFFICIO	35.000
R 601	COLLEGAMENTO TRA MICRO E MAINFRAME	39.000
	ELETTRONICA DI BASE E TECNOLOGIA	
201 A	CORSO DI ELETTRONICA FONDAMENTALE CON ESPERIMENTI	35.000
204 A	ELETTRONICA INTEGRATA DIGITALE	50.000
205 A	MANUALE PRATICO DI PROGETTAZIONE ELETTRONICA	35.000
200 A	SISTEMI DIGITALI: MANUTENZIONE, RICERCA ED ELIMINAZIONE GUASTI	28.500
GES262	TECNOLOGIE VLSI	70.000
GES390	ELETTRONICA INTEGRATA DIGITALE IL LIBRO DELLE SOLUZIONI	17.000
CE 411	LA FISICA DEI SEMICONDUTTORI	10.000
158 PC	ELETTRONICA DI BASE I FONDAMENTI DELL'ELETTRONICA ANALOGICA	55.000

GEJ 0006

CODICE	TITOLO	PREZZO
158 CC	ELETTRONICA DIGITALE VOL. 1 DALLE PORTE LOGICHE AI CIRCUITI INTEGRATI	55.000
158 DC	ELETTRONICA DIGITALE VOL. 2 DAI BUS AI GATE ARRAY	55.000
158 GC	ELETTROTECNICA ELETTROSTATICA ELETTROMAGNETISMO RETI ELETTR.	55.000
	ELETTRONICA: CIRCUITI E COMPONENTI	
601 B	TIMER 555	10.000
203 A	CIRCUITI INTEGRATI DIGITALI	10.000
612 P	MANUALE DEGLI SCR	28.000
613 P	MANUALE DI OPTOELETTRONICA	15.000
614 A	FIBRE OTTICHE	15.000
GE 403	JFET MOS E DATA BOOK	20.000
GE 404	TRANSISTOR DATA BOOK	32.000
GE 405	METODI DI PROTEZIONE CONTRO LE SOVRATENSIONI	17.000
CE 413	IL MANUALE DEGLI SCR E TRIAC	15.000
CE 421	MANUALE DEI FILTRI ATTIVI	29.000
CE 423	MANUALE DEI PLL PROGETTAZIONE DEI CIRCUITI	29.000
CE 425	MANUALE DEGLI AMPLIFICATORI OPERAZIONALI	29.000
CE 429	250 PROGETTI CON GLI AMPLIFICATORI DI NORTON	39.000
CE 431	MANUALE DEI CMOS	25.000
CE 485	IL COLLAUDO DELLE SCHEDE	18.000
BE 557	I TRASDUTTORI	43.000
	ELETTRONICA: APPLICAZIONI	
701 P	MANUALE PRATICO DEL RIPARATORE RADIO TV	29.000
705 P	IMPIEGO PRATICO DELL'OSCILLOSCOPIO	17.500
618 P	MISURE ELETTRONICHE E DIAGNOSI DEI GUASTI	34.500
708 P	MASTER TVC 1	30.000
709 P	MASTER TVC 2	30.000
615 P	PROGETTAZIONE DI SISTEMI DI ALTOPARLANTI	21.000
CE 427	L'ELETTRONICA A STATO SOLIDO	25.000
	ELETTRONICA: MICROPROCESSORI	
310 P	NANOBOOK Z80 VOL. 1	20.000
007 A	BUGBOOK VII	17.000
314 P	TECNICHE DI INTERFACCIAMENTO DEI MICROPROCESSORI	31.000
312 P	NANOBOOK Z80 VOL. III	25.000
320 P	MICROPROCESSORI DAI CHIPS AI SISTEMI	29.000
324 P	PROGRAMMAZIONE DELLO Z80 E PROGETTAZIONE LOGICA	21.500
326 P	Z80 PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLY	50.000
328 D	PROGRAMMAZIONE DELLO Z80	40.000
504 B	APPLICAZIONI DEL 6502	17.000
503 B	PROGRAMMAZIONE DEL 6502	35.000
505 B	GIOCHI CON IL 6502	19.500
342 A	CAPIRE I MICROPROCESSORI	10.000
G 220	8086-8088 PROGRAMMAZIONE	40.000
GY 265	ASSEMBLER PER IL 68000	70.000
CE 410	IMPIEGO DELLO Z80	23.000
158 HC	MICROPROCESSORI ARCHIT. PROGR. E INTERFAC. DEI MP DA 4 A 32 BIT	55.000
013 H	ASSEMBLER 6502	8.500
016 H	ASSEMBLER Z80	8.500
021 H	ASSEMBLER 68000	8.500
025 H	ASSEMBLER 8086-8088	8.500

CODICE	TITOLO	PREZZO
029 H	ASSEMBLER 80286	8.500
	AUTOMAZIONE	
208 A	CONTROLLORI PROGRAMMABILI	24.000
616 P	CONTROLLO AUTOMATICO DEI SISTEMI	29.500
GES251	STRUTTURA E FUNZIONAMENTO DEI CONTROLLI NUMERICI	29.000
GES252	CONTROLLI NUMERICI: PROGRAMMAZIONE E APPLICAZIONI	28.000
G 399	30 APPLICAZIONI DI CAD	29.000
G 401	CAD/CAM & ROBOTICA	28.000
CI 414	DAL CHIP ALLA ROBOTICA	15.000
GE 547	LA PROGETTAZIONE AUTOMATICA	32.000
	DIZIONARI ENCICLOPEDICI	
DS 498	FISICA	14.000
DS 499	MATEMATICA	14.000
DS 522	GEOLOGIA	14.000
DS 524	ELETTRONICA	14.000
DS 525	ASTRONOMIA	14.000
DS 526	CHIMICA	14.000
DS 527	RAGIONERIA GENERALE	14.000
DS 528	RAGIONERIA APPLICATA	14.000
DS 529	BIOLOGIA	14.000
DS 530	MECCANICA	14.000
DS 531	INFORMATICA	14.000
	ARGOMENTI VARI	
704 D	MANUALE PRATICO DI REGISTRAZIONE	10.000
706 A	COMUNICAZIONI RADIO IN MARE	18.000
800 H	FENDER, STORIA DI UN MITO	28.000
R 574	MANUALE DELLE STAMPANTI LASER	25.000
AQ 1861	AUTOMOBILI QUARTERLY N. 1	20.000
AQ 1872	AUTOMOBILE QUARTERLY N. 2	20.000
AQ 1873	AUTOMOBILE QUARTERLY N. 3	20.000
NQ 1861	NAUTICAL QUARTERLY N. 1	20.000
NQ 1862	NAUTICAL QUARTERLY N. 2	20.000
NQ 1863	NAUTICAL QUARTERLY N. 3	20.000
NQ 1874	NAUTICAL QUARTERLY N. 4	20.000
NQ 1875	NAUTICAL QUARTELY N. 5	20.000
	LIBRI PER RAGAZZI	
005 D	ENTRIAMO NEL CHIP: COME FUNZIONA E COSA PUÒ FARE	9.000
006 D	GIOCHI CON IL COMPUTER: COME FUNZIONANO, COME SI VINCE	9.000
003 D	ROBOT	9.000
007 D	PRIMI PASSI IN BASIC: UNA FACILE GUIDA PER SCRIVERE PROGRAMMI	9.000
008 D	CONOSCERE IL PERSONAL: COME LAVORA E COSA PUÒ FARE	9.000
009 D	COSTRUISCI PROGRAMMI DI ADVENTURE PER IL TUO COMPUTER	9.000
010 D	GIOCHI SPAZIALI	9.000
011 D	BATTAGLIE CON IL COMPUTER	9.000
018 D	IMPARIAMO A PROGRAMMARE: BASIC PER PRINCIPIANTI	9.000
002 D	INTRODUZIONE AL LINGUAGGIO MACCHINA	9.000
001 D	APPLICAZIONI PRATICHE DEL PERSONAL COMPUTER	9.000
013 D	COMPUTER GRAFICA: DALL'ANIMAZIONE AGLI ARCADE	9.000
014 D	BASIC È FACILE	9.000
015 D	TUTTO CIÒ CHE AVRESTE VOLUTO SAPERE SUL COMPUTER	9.000
016 D	PRATICA DEL BASIC	9.000
017 D	GIOCHI DI SPIONAGGIO: BRIVIDO E MISTERO	9.000

GEJ 0006

CODICE	TITOLO	PREZZO
019 D	MISTERO DELLA MONTAGNA D'ARGENTO	9.000
020 D	DIVERTIRSI CON IL PERSONAL	9.000
004 D	RIVOLUZIONE INFORMATICA	9.000
022 D	ESPANSIONI DEL PERSONAL COMPUTER	9.000
023 D	COMPUTER CON FANTASIA	9.000
026 D	ISOLA DEI SEGRETI	9.000
CU 001	COFANETTO USBORNE (N. 1)	45.000
CU 002	COFANETTO USBORNE (N. 2)	45.000
SOFTWARE E MANAGEMENT TOOLS		
CZ 469	GRAFIX - DISEGNARE CON IL PC (FLOPPY)	50.000
TP 606	CORSO AUTOISTRUZIONE LOTUS 1-2-3 (VERS. ITALIANA) F - MS DOS	90.000
TY 605	CORSO AUTOISTRUZIONE SUL SISTEMA MS DOS - FLOPPY	50.000
TY 640	TURBO PASCAL - LIBRERIA DI PROGRAMMI F - MS DOS	40.000
TP 643	CORSO AUTOISTRUZIONE LOTUS 1-2-3 (INGLESE) F - MS DOS	90.000
TP 608	BUDGET STRATEGICO (LOTUS 1-2-3) F - MS DOS	100.000
TP 614	GESTIONE DELLE COMMESSE DI PRODUZIONE - F - MS DOS	100.000
TP 623	CONTROLLO DELLE VENDITE (CON MULTIPLAN) F - MS DOS	100.000
TP 625	GESTIONE DEL PERSONALE (LOTUS 1-2-3) F - MS DOS	100.000
NOVITÀ GENNAIO '88		
BT 585	FIBRE OTTICHE	29.000
R 612	TURBO PROLOG	50.000
GYS546	ALGORITMI FONDAMENTALI	54.000
GY 618	SISTEMI ESPERTI	28.000
GE 567	80286 ARCHITETTURA E PROGRAMMAZIONE	58.000
GY 626	IL MANUALE DEL PASCAL	42.000
PP 611	GUIDA ALL'USO PROFESSIONALE REFLEX	55.000
PP 636	MANUALE DI WORD	70.000
CC 658	GRAFICA E SUONO PER C64 - 64PC - C128 - FLOPPY	35.000
CC 657	MANUALE DEL COMMODORE C64 - 64PC - C128 - FLOPPY	
NOVITÀ FEBBRAIO '88		
GY 616	DEBUGGING C	55.000
PP 594	GUIDA ALL'USO PROFESSIONALE DI LOTUS 1-2-3	50.000
CC 627	AMIGA 500	60.000
R 609	SOLUZIONI AVANZATE PER IL PROGRAMMATORE	60.000
CC 656	DALL'XT AL PS/2	29.000
044 T	MS DOS	14.500
045 T	PC DOS	14.500
BE 578	MANUALE DI ELETTRONICA	38.000
R 628	MICROSOFT OS/2	50.000
AQ1874	AUTOMOBILE QUARTERLY N. 4	20.000

* L'Editore si riserva di modificare i prezzi di copertina in qualsiasi momento.

GEJ 0006/A

Commodore 128 oltre il manuale

Rita Bonelli

Scopo di questo libro è fornire un aiuto per un approccio sistematico all'apprendimento della programmazione, più alcune notizie aggiuntive sul sistema, non contenute nei due manuali venduti insieme al calcolatore. Si è evitato di ripetere i contenuti dei manuali e si raccomanda quindi ai lettori di tenerli sempre a portata di mano, per ricercare quelle notizie che non sono state volutamente riportate.
Attualmente il mercato offre dei calcolatori molto potenti ad un prezzo abbastanza accessibile e l'utente si trova a contatto con questi strumenti senza aver avuto occasione prima di occuparsi dell'argomento. Di conseguenza, il libro non vuole essere un trattato di informatica, ma piuttosto contribuire in un modo "globale" alla conoscenza del calcolatore. Il lettore potrà ricevere un aiuto nell'apprendimento dell'uso della sua macchina e degli stimoli per approfondire alcuni argomenti.
Il modo più soddisfacente ed immediato per avvicinarsi all'informatica è quello di avere a disposizione un calcolatore, soprattutto se è facile da usare come un personal e, in particolare, come il Commodore 128. Il libro tratta prevalentemente la programmazione in BASIC 7.0 e in Assembler nel modo di funzionamento 128 oltre a presentare alcuni pacchetti di programmi in modo CP/M. Nel modo 128 viene presentata la grafica con video sia a 40 che a 80 colonne.

- Programmazione
- BASIC 7.0
- Utilizzo della memoria
- Assembler
- Linguaggio macchina
- Modo CP/M

GRUPPO EDITORIALE JACKSON

L. 29.000 Cod. CC322 ISBN 88-7056-426-6